ACELERACIÓN ENERGÉTICAMENTE EFICIENTE DE APLICACIONES CIENTÍFICAS DE GRAN ESCALA SOBRE ARQUITECTURAS HETEROGNÉAS

JOHN ANDERSON GARCÍA HENAO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
MAESTRÍA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

ACELERACIÓN ENERGÉTICAMENTE EFICIENTE DE APLICACIONES CIENTÍFICAS DE GRAN ESCALA SOBRE ARQUITECTURAS HETEROGNÉAS

JOHN ANDERSON GARCÍA HENAO

Trabajo de investigación para optar al título de
Magister en Ingeniería de Sistemas e Informática

Director
CARLOS JAIME BARRIOS HERNÁNDEZ, PhD.
Profesor de la Escuela de Ingeniería de Sistemas
Universidad Industrial de Santander

Codirector
PHILIPPE OLIVIER ALEXANDRE NAVAUX, PhD.
Profesor del Instituto de Informática
Universidade Federal do Rio Grande do Sul

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
MAESTRÍA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

Agradecimientos

- A los profesores Carlos Jaime Barrios H. y Philippe Olivier A. Navaux,
- A los compañeros Víctor Eduardo Martínez A. y Esteban de Jesús Hernández B.,
- A mi cordada de vida Zayne Milena Roa Díaz,
- Al Grupo de Investigación Supercomputación y Cálculo Científico - SC3 de la UIS,
- Al Grupo de Investigación Parallel and Distributed Processing Group - GPPD de la UFGRS.

Dedicatoría

Este trabajo de investigación esta dedicado
a mi familia quien constituye un escenario de unión y alegría.
Especialmente a mi madre quien ha luchado con su espíritu del alma del corazón
para ayudarme a continuar con mi camino
y a mi padre quien me fortalece día a día desde la distancia.

Índice general

Índice de figuras

Índice de cuadros

Índice de Anexos

ACELERACIÓN ENERGÉTICAMENTE EFICIENTE DE APLICACIONES CIENTÍFICAS DE GRAN ESCALA SOBRE ARQUITECTURAS HETEROGNÉAS[1]

JOHN ANDERSON GARCÍA HENAO[2]

Palabras Clave: Eficiencia energética, enerGyPU, esquema *energy-aware* AEE, monitor de rendimiento y consumo de potencia, programación basada en tareas.

Resumen

La computación de alto rendimiento es hoy una herramienta fundamental para la investigación científica y la competitividad industrial. Por lo que requiere construir grandes sistemas computacionales con mayor rendimiento, los cuales ven limitada su capacidad por el consumo energético y la subutilización de recursos computacionales en aplicaciones que realizan una mala distribución de tareas, es así como maximizar el rendimiento por *watt* y realizar un buen mapeo de tareas es uno de los principales retos para construir la siguiente generación de sistemas exascale.

Este trabajo de investigación presenta una comparación de consumo de energía en tiempo de ejecución, entre arquitecturas heterogéneas tipo CPU-GPU con diferentes frecuencias de reloj y número de CUDA@cores dentro de un mismo chip, al usar StarPU, para portabilizar y balancear la carga de trabajo de aplicaciones científicas. Como caso de estudio se seleccionó Ondes3D, una simulación de la propagación de ondas sísmicas para analizar fuertes movimientos en superficies de la Tierra. Adicionalmente, se seleccionó el LINPACK Benchmark (HPL), para colectar datos del problema de la subutilización de recursos computacionales sobre arquitecturas heterogéneas. Se construyó enerGyPU un monitor de rendimiento y consumo de potencia, para evaluar y caracterizar los factores computacionales que regulan la eficiencia energética sobre nodos heterogéneos con multiples GPU.

Asimismo, se diseño un esquema integrado llamado Aceleración Energéticamente Eficiente (AEE), el cual utiliza: frameworks de balanceo de carga (ej. StarPU), para el mapeo de tareas sobre los recursos computacionales; seguidamente, usa enerGyPU para la captura de métricas de rendimiento computacional y consumo de potencia, para así, caracterizar la aplicación y la arquitectura computacional; por ultimo utiliza el sistema de predicción AEE, para obtener la combinación de recursos computacionales que maximizan la eficiencia energética en aplicaciones que se ejecuten sobre arquitecturas heterogéneas tipo CPU-GPU.

[1]Trabajo de Investigación para optar al título de Magister en Ingeniería de Sistemas e Informática.

[2]Escuela de Ingeniería de Sistemas e Informática, Facultad de Ingenierías Físico-mecánicas en la Universidad Industrial de Santander. Codirector: Philippe Olivier Alexandre Navaux, Phd. Profesor del Instituto de Informática, Universidade Federal do Rio Grande do Sul. Director: Carlos Jaime Barrios Hernández, PhD. Profesor de la Escuela de Ingeniería de Sistemas, Universidad Industrial de Santander.

EFFICIENTLY ENERGETIC ACCELERATION FOR SCIENTIFIC COMPUTATIONS OF LARGE-SCALE ON HETEROGENEOUS ARCHITECTURES[3]

JOHN ANDERSON GARCÍA HENAO[4]

Key Words: EEA prediction system, Energy efficiency, enerGyPU, monitor of performance and power, task-based programming.

Abstract

High performance computing is now an essential tool for scientific research and industrial competitiveness. Therefore, it requires building large computer systems with higher performance, which are limited in their capacity for energy consumption and underutilization of computing resources in applications that perform poor distribution of tasks, so as maximize performance per watt and make good mapping tasks is one of the main challenges to build the next generation of exascale systems.

This research presents a comparison of energy consumption at runtime, among heterogeneous architectures CPU-GPU type with different clock frequencies and number of CUDA@cores within a single chip, using StarPU to portability and load balancing work of scientific applications. As a case study was selected Ondes3D, a simulation of the propagation of seismic to analyze strong movements in the earth wave surfaces. Additionally, it was selected the LINPACK Benchmark (HPL), to collect data of the problem of underutilization of computing resources on heterogeneous architectures. It was built enerGyPU monitor performance and power consumption, to evaluate and characterize the computational factors that regulate energy efficiency on heterogeneous nodes with multiple GPU.

Also this research designed an integrated scheme called Energy Efficient Acceleration (AEE), in which can be used by load balancing frameworks (eg. StarPU) for mapping tasks on the computational resources; then use enerGyPU for capturing metrics of performance and power consumption in order to characterize the application and computing architecture; finally prediction system uses the scheme AEE, for the predict the combination of computational resources that maximize energy efficiency of applications running on heterogeneous architectures CPU-GPU type.

[3]Degree work to obtain the title of Master of Science in Systems Engineering and Informatics in Industrial University of Santander.

[4]Escuela de Ingeniería de Sistemas e Informática, Facultad de Ingenierías Físico-mecánicas. Advisor of Research: Philippe Olivier Alexandre Navaux, Phd. Professor at Instituto de Informática,Federal University of Rio Grande do Sul - UFRGS. Director of Research: Carlos Jaime Barrios Hernández, PhD. Professor at Systems Engineering and Informatics School, Industrial University of Santander - UIS.

Introducción

La ciencia, la ingeniería y la industria cada vez más requieren de la computación de alto rendimiento para procesar modelos computacionales, simulaciones y predecir el comportamiento de problemas dinámicos, en donde realizar estos experimentos son imposibles, peligrosos o excesivamente costosos. La supercomputación y los próximos sistemas *exascale* permiten encontrar soluciones con mayor precisión y realizar análisis con cantidades masivas de datos, permitiendo avances cuantitativos en las diferentes áreas de la ciencia y la tecnología. Como lo describe el reporte de computación científica *exascale*, diseñado por el departamento de energía de U.S. [1], donde la supercomputación puede ampliar las fronteras en: la adaptación a los cambios climáticos regionales, tales como el aumento del nivel del mar, la sequía y las inundaciones, los patrones de clima severo; el modelado y el descifrado del cerebro humano; el diseño, control y fabricación de materiales avanzados; diseños innovadores para la energía sustentable, tales como baterías, catalizadores y los biocombustibles; la eficiencia y la seguridad del sector de la energía nuclear. Sin embargo el consumo de potencia es la principal limitante para el sostenimiento de estos supercomputadores y la construcción de la siguiente generación de sistemas *exascale*.

El consumo de potencia aumenta cuando los procesadores, coprocesadores y aceleradores gráficos incrementan la velocidad y/o el número de cores dentro de un mismo chip. Por ejemplo la NVIDIA® Tegra® k1 tiene un consumo de potencia de $15\,Watts$ (con $852MHz$ de frecuencia de reloj y 192CUDA© *cores*); en comparación con la NVIDIA® Tesla® k40 que tiene un mayor rendimiento computacional y un mayor consumo de potencia de $235\,Watt$ (con $875MHz$ de frecuencia de reloj y 2880CUDA© *cores*). Por lo tanto el gradiente de rendimiento computacional y consumo de potencia, varía dependiendo de la arquitectura y el diseño del circuito (numero de *cores*, frecuencia de reloj, tamaño de memoría y ancho de banda), como lo describe Hong et al. en [2]. Sin embargo, mapear la granularidad de tareas de una aplicación para lograr un alto rendimiento con un menor consumo de potencia para una determinada arquitectura es aún más difícil.

Por otra parte, la programación paralela utiliza diferentes modelos de programación para explotar el paralelismo dependiendo la arquitectura de la maquina, única instrucción múltiples hilos (SIMT) para las NVIDIA® Tesla® GPU o única instrucción múltiples datos (SIMD) para las Intel® Xeon® Phi. [3] Sobre las Intel® Xeon® Phi, el número de *cores*, *threads* por *core* y el nivel de afinidad es un aspecto critico para lograr un mayor rendimiento computacional y conocer los factores que pueden ser utilizados para minimizar el consumo de potencia. En las Nvidia® Tesla® GPU, el *kernel*, el *grid* y el numero de *threads* por bloque; el *streaming multiprocessor*, la frecuencia de reloj, la frecuencia de memoría, el uso de memoria y el ancho de banda es un aspecto principal para obtener un mayor rendimiento y bajo consumo de potencia. De esta manera los datos de entrada, la arquitectura y la complejidad de la carga de trabajo son un parámetro clave para determinar el número de recursos computacionales a utilizar y generar la granularidad de tareas a ser mapeadas en los supercomputadores.

Por consiguiente, este trabajo de investigación presenta una comparación de consumo de energía en tiempo de ejecución, entre arquitecturas heterogéneas tipo CPU-GPU con diferentes frecuencias de reloj y número de CUDA© *cores* dentro de un mismo *chip*, al usar StarPU©, para portabilizar y balancear la carga de trabajo de aplicaciones científicas. Como caso de estudio se seleccionó Ondes3D©, una simulación de la propagación de ondas sísmicas para analizar fuertes movimientos en superficies de la Tierra. Adicionalmente, se construyó enerGyPU un monitor de rendimiento y consumo de potencia, para centralizar y automatizar la captura de factores en tiempo de ejecución, como (la frecuencia de reloj, el uso de memoria y el ancho de banda). El cual gráfica la información vía secuencia de datos, tablas estadísticas, gráficos de barras y muestra los resultados en términos de eficiencia energética por cada ejecución de una aplicación. Como caso de estudio se seleccionó una carga de trabajo sintética llamada Highly-Parallel LINPACK® (HPL) Benchmark, un problema general de matriz densa para realizar la evaluación del rendimiento computacional y consumo de potencia sobre GUANE[5]. El HPL es ampliamente usado por el Top500 y el Green500 en la evaluación de rendimiento per watt sobre supercomputadores. Por lo tanto, este trabajo de investigación propone un esquema integrado *energy-aware* llamado Aceleración Energéticamente Eficiente (AEE), el cual utiliza: *frameworks* de balanceo de carga (ej. StarPU©), para el mapeo de tareas sobre los recursos computacionales; seguidamente, usa enerGyPU para la captura de métricas de rendimiento computacional y consumo de potencia, para así, caracterizar la aplicación y la arquitectura computacional; por ultimo utiliza el sistema de predicción AEE, para obtener la combinación de recursos computacionales que maximizan la eficiencia energética en aplicaciones que se ejecuten sobre arquitecturas heterogéneas tipo CPU-GPU.

Este documento esta organizado de la siguiente forma: El capitulo #1 presenta el contexto de investigación. El capitulo #2 define los fundamentos teóricos: las plataformas y arquitecturas computacionales usadas, modelos de programación paralela y *frameworks* de programación basado en tareas. En el capitulo #3 se describen las métricas de consumo de energía utilizadas y se presenta un análisis de eficiencia energética y portabilidad de Ondes3D©. En el capitulo #4 se describe enerGyPU para la evaluación de rendimiento computacional y consumo de potencia. En el capitulo #5 se introduce el esquema integrado de Aceleración Energéticamente Eficiente AEE y el sistema AEE para la predicción de tiempo y potencia. Finalmente el capitulo #6 presenta la discusión y las conclusiones de este trabajo de investigación en ciencias de la ingeniería de sistemas e informática.

[5]GUANE-1 (GpU AdvaNced Environment for scientific computing) está situado en el Centro de Supercomputación y Cálculo Científico de la Universidad Industrial de Santander (SC3-UIS), localizado en el Parque Tecnológico de Guatiguará (Piedecuesta), un municipio del area metropolitana de Bucaramanga, Santander - Colombia. ver http://www.sc3.uis.edu.co/

Capítulo 1

Contexto de Investigación

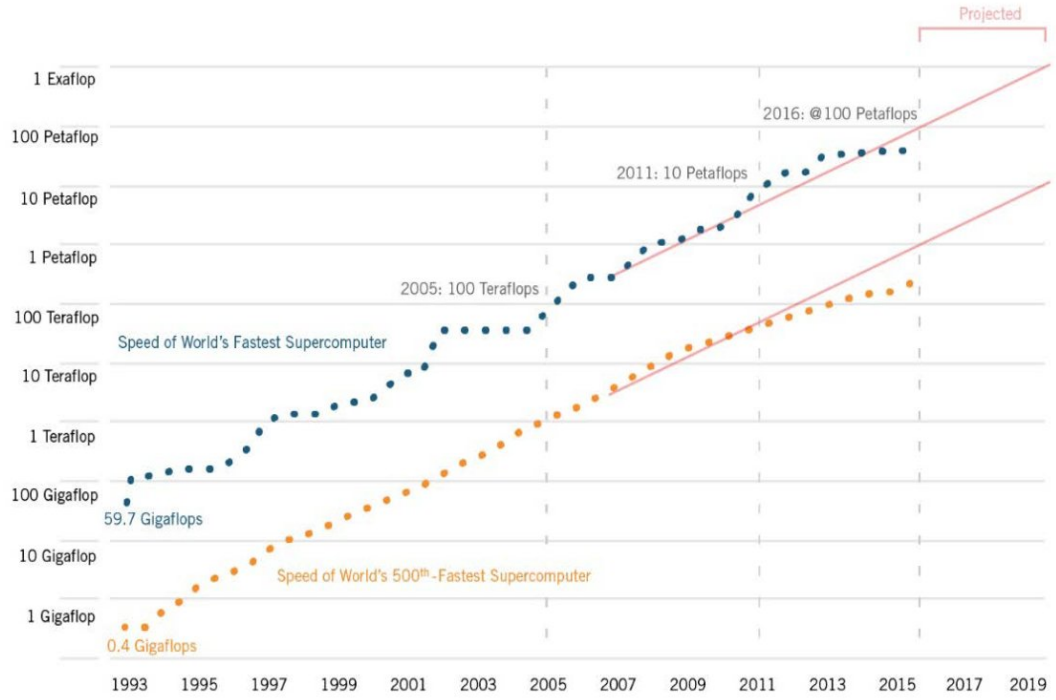
1.1. Planteamiento del Problema

1.1.1. Identificación del Problema

Desde 1993 se ha estado construyendo sistemas computacionales de alto rendimiento, teniendo en cuenta únicamente el máximo rendimiento alcanzado (*Rmax*), medido por la cantidad de operaciones punto flotante por segundo (*FLOP/s*) de acuerdo a la lista Top500 como se muestra en la Figura 1.1. En la cual, los supercomputadores han aumentado exponencialmente su consumo energético en proporción a la cantidad de recursos computacionales utilizados para obtener un alto rendimiento, durante las últimas dos décadas. En particular, el supercomputador con mayor *Rmax* alcanzado en la lista Top500 de noviembre de 2015 es: la Tianhe-2 con 3'120.000*cores* de procesamiento, para lograr un rendimeinto computacional de 33,8*petaFLOP/s* con un consumo de potencia de 17,8*megaWatt*, alcanzando un rendimeinto por *watt* de 1,9*gigaFLOPS/Watt* [4]. Sin embargo, los supercomputadores listados en la top500 del 2015, no cumplen con las condiciones necesarias de rendimiento computacional y consumo de potencia para construir un supercomputador a nivel Exascale para el año 2020, las cuales han sido propuestas en el estudio *Technology challenges in Achieving Exascale Systems* por DARPA en [5]. En síntesis, este estudio tiene como objetivo construir un supercomputador que alcance un rendimiento computacional de 1*exaFLOP* con un consumo de potencia de 20*megaWatt*, para lograr una eficiencia de rendimiento por *Watt* de 50*gigaFLPOS/Watt*.

Por otro lado, cuando se ejecutan aplicaciones científicas de gran escala que han sido desarrolladas a lo largo del tiempo, la velocidad en la que estos supercomputadores procesan los modelos es de 5 % a 8 % del rendimiento máximo *Rmax*, cuando se ejecutan aplicaciones muy paralelas [6]. Esta subutilización de los recursos computacionales y a su vez incremento en el consumo energético por parte de las aplicaciones se debe a la evolución del hardware. Tradicionalmente el crecimiento computacional ha sido impulsado por el aumento de frecuencia de reloj, la miniaturización y el aumento de transistores en los procesadores. Por consiguiente las aplicaciones pueden ser aceleradas a través del

Figura 1.1: Rendimeinto computacional por año de la lista top500, entre 1993-2015.



Fuente **Stephen J. Ezell.** The Vital Importance of High-Performance Computing to U.S. Competitiveness, 2016

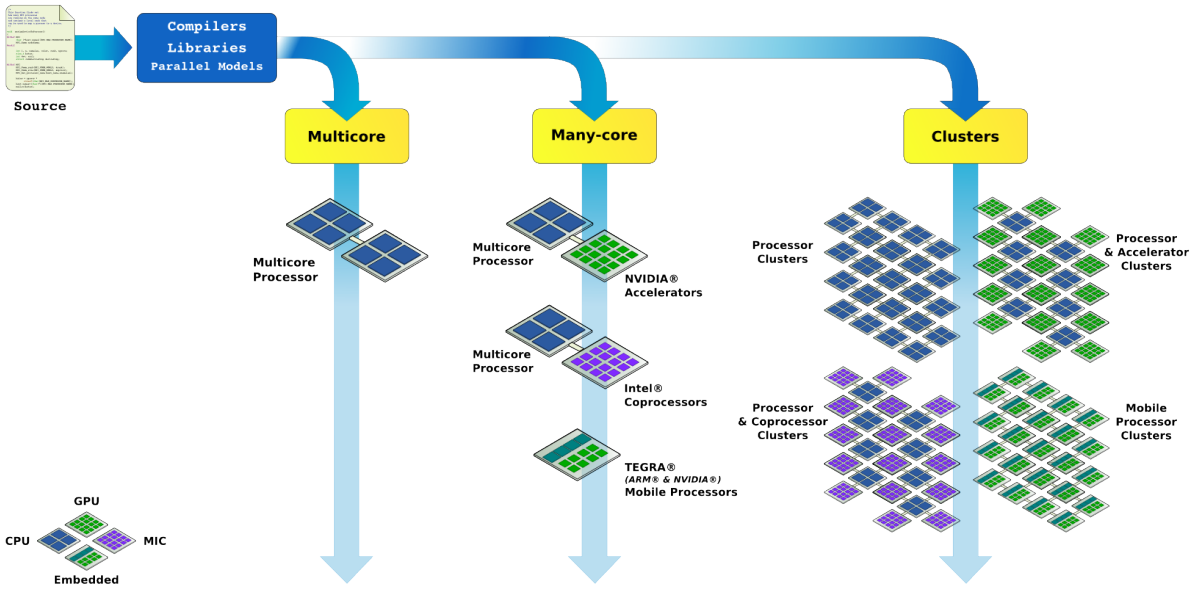
paralelismo a nivel de instrucción, es decir, de acuerdo a la cantidad de operaciones que se puede ejecutar simultaneamente. Hoy en día, la tendencia de aumento de frecuencia de reloj es cada vez mas difícil de mantener, debido a las limitaciones térmicas en los procesadores. Para resolver este problema la industria y la comunidad de investigación, ha empezado a utilizar una mayor área para implementar la mayor cantidad de *cores* dentro de un mismo procesador, surgiendo así los procesadores *multicore*, coprocesadores y aceleradores *manycore*. Por lo que, si una aplicación tiene suficiente paralelismo, estos chips con más *cores* individuales y menor frecuencia de reloj dentro de un mismo *chip* pueden alcanzar aceleraciones con menor consumo de energía. Es importante destacar que existe gran variedad de arquitecturas heterogéneas y cada una tiene características particulares que requieren ser comprendidas para balancear la carga de trabajo y minimizar el consumo energético.

Como consecuencia la programación paralela aplicada en grandes sistemas computacionales se enfrenta a dos problemas: uno es el incremento de consumo de potencia en supercomputadores directamente relacionado con la cantidad de recursos computacionales usados para obtener un alto rendimiento, y el otro problema es la subutilización de estos recursos computacionales por parte de aplicaciones que realizan una mala distribución de tareas. Para contribuir con la solución de estos problemas, este trabajo de investiga-

ción busca seleccionar una configuración de recursos computacionales óptima y realizar un buen mapeo de granularidad de tareas, siendo este uno de los principales retos para construir la siguiente generación de Sistemas Exascale.

Una observación clave del estudio *Technology challenges in Achieving Exascale Systems*, realizado por DARPA [5] es; que puede ser más facil resolver el problema de consumo de potencia con base a la computación, que resolver el problema de transportar los datos de un sitio a otro dentro del mismo chip, transportar los datos entre diferentes chips que están estrechamente a complados en un mismo nodo o transportar los datos entre diferentes *racks* en lados opuestos de una gran sala de máquinas (Ver Figura 1.2). En este orden, las aplicaciones de gran escala requieren de una evaluación de rendimeinto y medición del consumo de potencia por cada arquitectura computacional, para analizar multiples pruebas de ejecución utilizando diferentes combinaciones de parametros y observar los factores clave que determinan la eficiencia energética en terminos de energía por computación, energía por dato a transportar, energía por acceso a memoria o energía por unidad de almacenamiento.

Figura 1.2: Organización de procesadores y coprocesadores en diferentes maquinas computacionales.



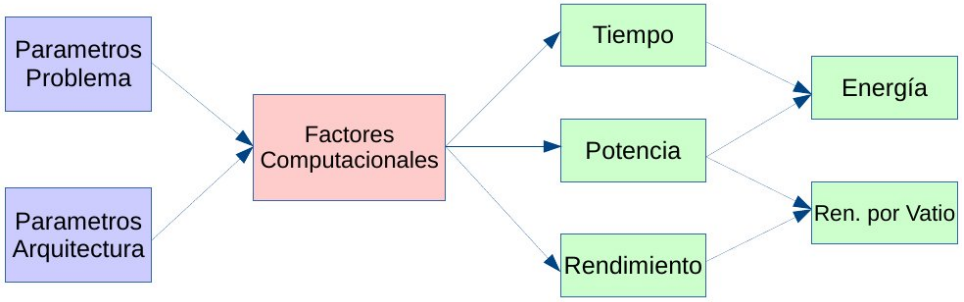
1.1.2. Pregunta de Investigación

¿Cuál es la distribución de tareas a ser mapeadas sobre los núcleos de una arquitectura heterogénea tipo CPU-GPU, que determina un eficiente aceleramiento computacional y

bajo consumo energético durante el tiempo de ejecución de aplicaciones científicas de gran escala?

Esta pregunta caracteriza el trabajo de investigación con estudios de alcance explicativo [7]. Por lo tanto se busca explicar los factores computacionales que regulan el consumo de energía en supercomputadores con arquitecturas heterogéneas y maximizan el rendimiento computacional en la ejecución de aplicaciones científicas de gran escala (Ver Figura 1.3).

Figura 1.3: Factores computacionales que determinan una aceleración eficientemente energética.



1.2. Objetivos

Objetivos General

- Determinar la distribución de tareas de aplicaciones científicas de gran escala sobre una arquitectura heterogénea tipo CPU-GPU, para obtener un eficiente aceleramiento computacional y bajo consumo energético durante el tiempo de ejecución.

Objetivos Específicos

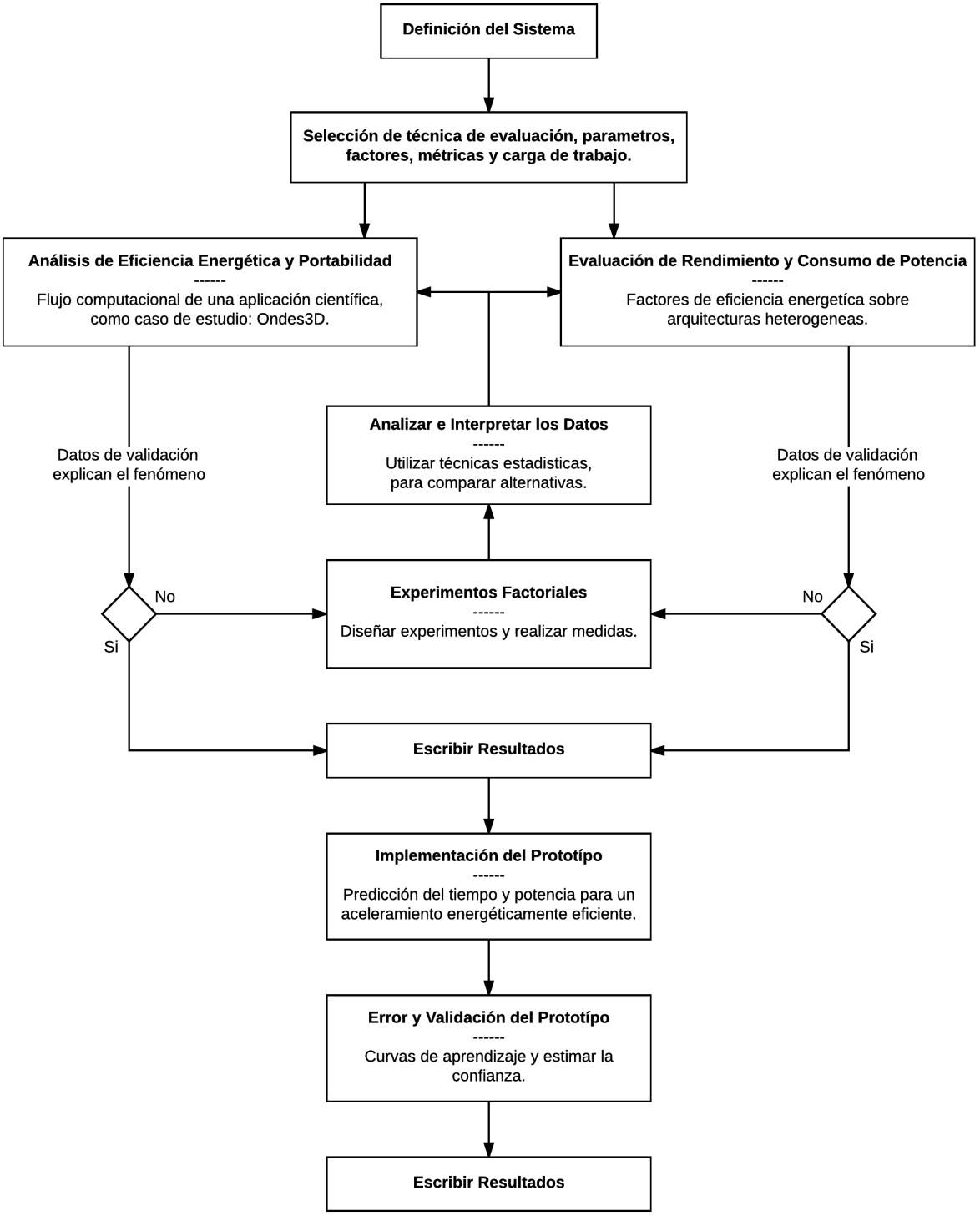
- Seleccionar métricas, parámetros y cargas de trabajo, para capturar el rendimiento computacional y el consumo energético en arquitecturas heterogéneas tipo CPU-GPU.
- Analizar el flujo computacional de una aplicación científica de gran escala seleccionada, para proponer un mecanismo de aceleración energéticamente eficiente.
- Evaluar cargas de trabajo sintéticas *benchmark* sobre configuraciones de arquitectura CPU-GPU, para determinar la distribución de tareas que permite un eficiente aceleramiento computacional y consumo energético durante el tiempo de ejecución.

- Plantear un mecanismo que distribuya las tareas de un problema matricial de gran escala y regule la frecuencia de procesamiento, para acelerar el rendimiento computacional de las aplicaciones y reducir el consumo de energía en arquitecturas heterogéneas tipo CPU-GPU.

1.3. Metodología Aplicada

La metodología de investigación que se adopto en este trabajo de investigación, es la relación entre los dos problemas que enfrenta la programación paralela aplicada en grandes sistemas computacionales, los objetivos planteados y la Metodología para la evaluación de rendimiento computacional, propuesta por Raj Rain en [8]. En la ver Figura 1.4 se establece la selección de técnicas de evaluación, parametros, factores, métricas que se utilizaron, para: analizar la eficiencia energética y portabilidad de una aplicación científica de gran escala sobre arquitecturas heterogéneas, y así colectar datos que solucionan el problema de la subutilización de recursos computacionales sobre arquitecturas heterogéneas. Así mismo se recogieron datos, para evaluar y caracterizar los factores computacionales que regulan la eficiencia energética sobre nodos heterogéneos con multi-GPU. Por ultimo se diseña e implementa un prototipo de predicción del tiempo y la potencia para un aceleramiento energéticamente eficiente.

Figura 1.4: Metodología de Investigación



Capítulo 2

Fundamentos Teóricos

2.1. Plataformas Computacionales

En esta sección se describe la plataforma y la arquitectura computacional utilizadas en este proyecto de investigación, para analizar la ejecución de aplicaciones y evaluar el tiempo de ejecución, el rendimiento computacional, el consumo de potencia, el consumo de energía y el rendimiento por *Watt*.

2.1.1. Cluster Computacional con Nodos Tipo CPU-GPU

GUANE es la plataforma computacional seleccionada como caso de uso en supercomputadores con nodos de gran capacidad computacional, para medir y determinar el consumo de energía empleada durante el procesamiento de aplicaciones de gran escala que utilizan paralelamente *multicore* y multiGPU. GUANE es un *cluster* computacional con una arquitectura heterogénea tipo CPU-GPU, compuesto por un *rack* de 16 nodos *ProLiant SL390s* con tres configuraciones distintas, como se muestra en el Cuadro 2.1.

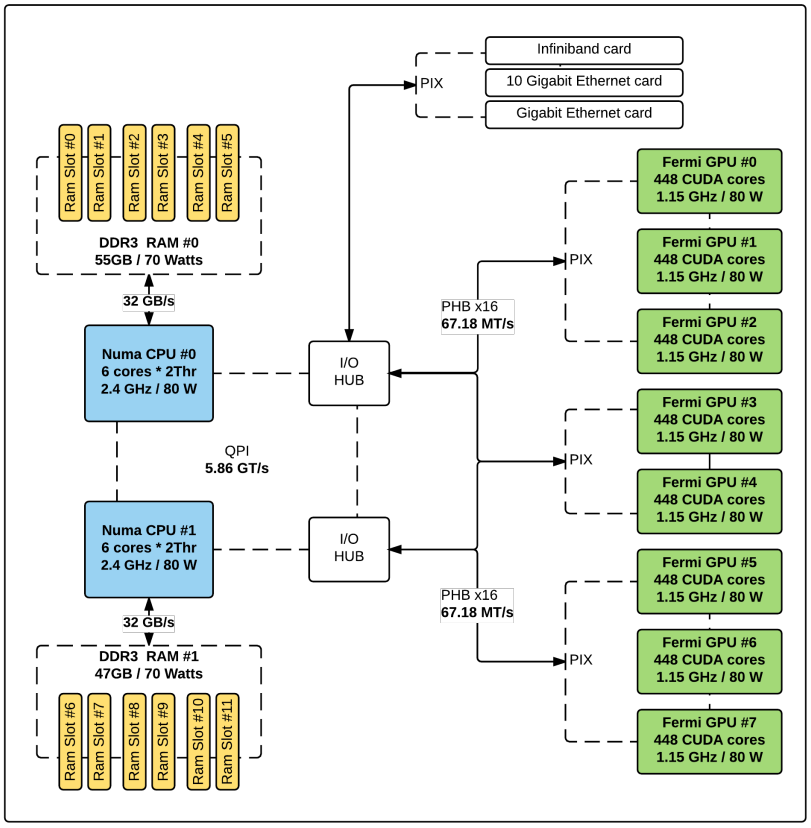
La Figura 2.1 representa la organización y la forma de interconectar los recursos computacionales en cada uno de los 16 nodos *ProLiant SL390s* de GUANE, la cual cambia el modelo de la CPU y la GPU dependiendo la configuración del nodo seleccionado. Así mismo muestra algunos factores computacionales como la frecuencia de reloj y el número de *cores*, que se pueden controlar en diferentes instantes de tiempo para maximizar el rendimiento por *Watt*.

Las configuraciones de nodos tipo A y B cuentan con 2 procesadores Intel® Xeon® E5645, con un total de 24*threads* lógicos para HyperThreading® y 12*cores* físicos de los cuales 8*cores* se utilizan para el gerenciamiento de transferencia de datos entre la memoria principal hasta la memoria gráfica *GDDR5* de las 8 GPU; los 4*cores* restante se pueden aprovechar para el procesamiento paralelo *multicore* y multiGPU. Por consiguiente la configuración de nodos tipo C con 2 procesadores Intel® Xeon® E5640, tiene un total de 16*threads* lógicos para HyperThreading® y 8*cores* físicos exactos para el ge-

Cuadro 2.1: Características computacionales de GUANE.

Configuraciónes	**A**	**B**	**C**
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Microarchitecture	Sandy Bridge	Sandy Bridge	Sandy Bridge
Processor by node (#)	2	2	2
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
Microarchitecture	Fermi	Fermi	Fermi
GPUs by node (#)	8	8	8
CUDA core (#)	448	448	448
Memory DDR4 (GB)	104	104	104
SAS disk (GB)	200	200	200
Gigabit Ethernet (Gbps)	10	10	10
InfiniBand (IB)	1	1	1

Figura 2.1: Arquitectura computacional de cada nodo de GUANE

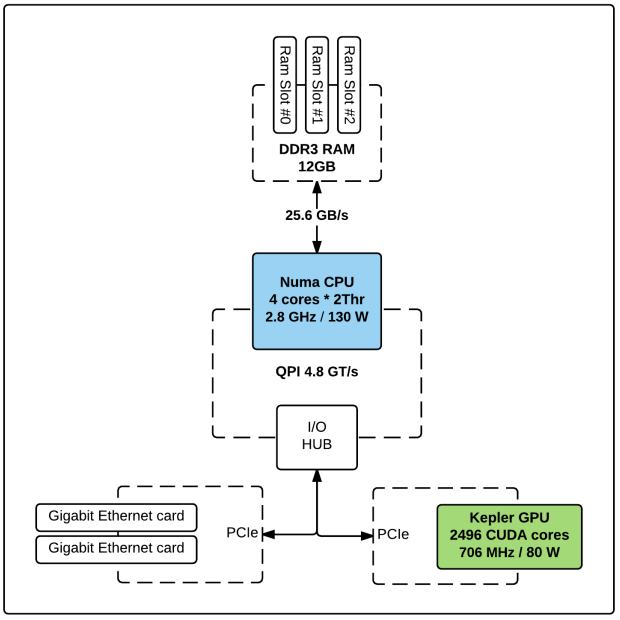


renciamiento de las 8 GPU; permitiendo procesamiento paralelo *multicore* y multiGPU, solo en casos donde se utilice menos de 8 GPU o aplicando técnicas de programación dinámica y *overlapping*, para aprovechar los mismos *cores* en tareas de gerenciamiento y procesamiento.

2.1.2. Servidor Computacional con NVIDIA® Tesla^{TM} GPU

Se utilizo un servidor que puede ser visto como una arquitectura commodity-based. Esta maquina tiene: un procesador Intel Core^{TM} i7-930@2.80GHz (4 *cores* fisicos) y una aceleradora NVIDIA Tesla^{TM} K20c with 5GB con una GDRAM5 de (2496@CUDA*cores*) como se muestra en la Figura reffig:Desktop-NodeArchitecture.

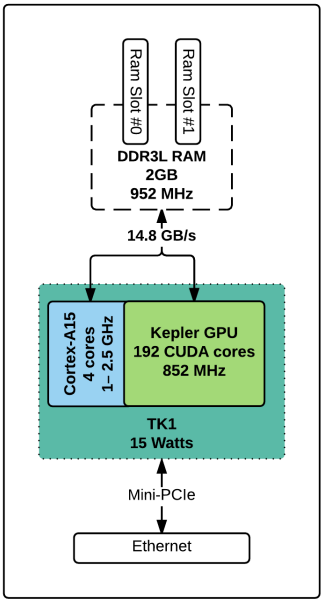
Figura 2.2: Arquitectura de un computador de esritorio



2.1.3. Computadora Embebida con NVIDIA® TEGRA® K1^{TM}

La otra plataforma que se utilizó es la Jectson TK1, desarrollada por NVIDIA, la cual trae un porcesador *quad-core* ARM R Cortex-A15 y un Streaming Multiprocessor Kepler con 192@CUDA*cores* en un mismo *chip* y tiene una memoria RAM con solo 2 GB de memoria compartida enre la CPU y la GPU.

Figura 2.3: Arquitectura de una placa Jetson Tk1



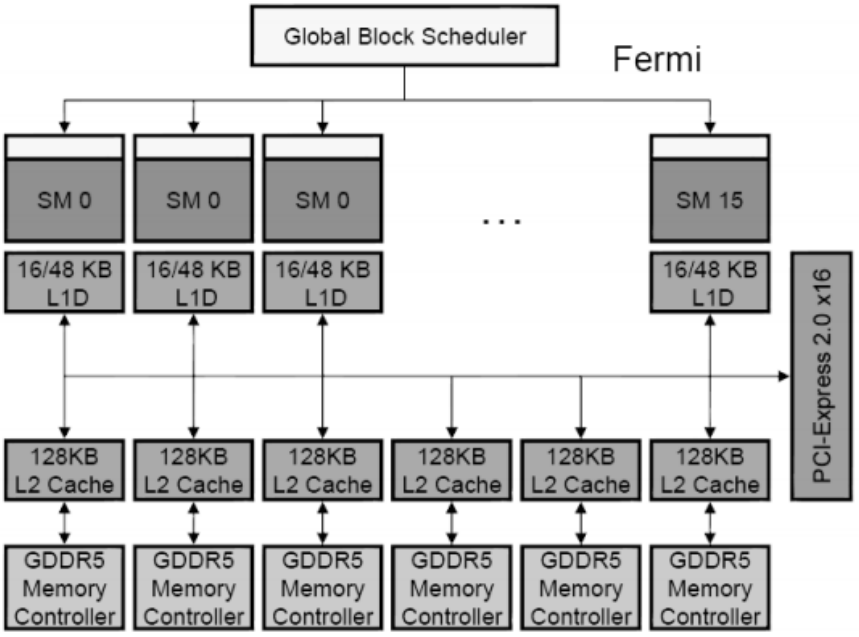
2.2. Arquitecturas Computacionales

2.2.1. Microarquitectura NVIDIA® Fermi^{TM}

NVIDIA® ha desarrollado diferentes modelos de tarjetas gráficas de propósito general, aumentando el número de transistores por unidad a la vez que incrementa la cantidad de CUDA© *cores*, para maximizar la capacidad de computo en las GPU. Por consiguiente, se tiene diferentes arreglos de CUDA© *cores* (*SM: Streaming Multiprocessor*) con diferentes tipos de microarquitectura, como: Tesla^{TM} con 8CUDA© *cores* por cada *SM*; Fermi^{TM} con 32CUDA© *cores* por cada *SM*; Kepler^{TM} con 192CUDA© *cores* por cada *SMX*; Maxwell^{TM} con 128CUDA© *cores* por cada *SMM*; y Pascal^{TM} que regresa con 64CUDA© *cores* por cada *SMP*, dividido en dos bloques de procesamiento con 32*SP*CUDA© *cores*, un *buffer* de instrucciones, un *warp scheduler*, de acuerdo al reporte técnico de *NVIDIA Corporation* en [9].

Las 128 GPU de GUANE estan dividas en los modelos NVIDIA® Tesla® M2050 y M2075, con una microarquitectura Fermi^{TM}. Las cuales traen 512CUDA© *cores* distribuidos en 16*SM* con 32*cores* por cada uno, como se muestra en la Figura 2.4. Esta fammilia de GPU tiene seis particiones de memoria de 64*bits*, una interfaz de memoria de 384*bits*, soportando hasta un total de 6*GByte* de GDDR5 de memoria DRAM. La interfaz de conexion de la CPU a la GPU se realiza via PCI-Express. El calendizador global distribuye los bloques de *threads* hasta el calendizador de *threads* del Streaming Multiprocessor.

Figura 2.4: Arquitectura computacional de cada nodo de GUANE



2.3. Modelos de Programación Paralela

Muchos lenguajes y modelos de programación paralela se han propuesto en las últimas décadas. Los ampliamente utilizados son los de *Message Passing Interface* (MPI) para clusters computacionales escalables y OpenMP para sistemas con multiprocesador y memoria compartida. MPI es un modelo de programación en el que los nodos de computación de un *cluster* no comparten la memoria, todo el intercambio de datos y la interacción se deben hacer a través del paso de mensajes. Las aplicaciones escritas en MPI han sido conocidas por ejecutarse con éxito en *clusters* computacionales con más de 100.000 nodos. Sin embargo, la cantidad de esfuerzo requerido para portar una aplicación al modelo de programación MPI, puede ser extremedamente alto debido al esfuerzo requerido para compartir la memoria a través de los nodos de computación, transportar los datos y realizar balanceo de carga. CUDA, por otro lado, proporciona memoria compartida para la ejecución en paralelo en la GPU y hacer frente a esta dificultad. En cuanto a la comunicación de CPU y GPU, CUDA proporciona una capacidad de memoria compartida muy limitada entre 6GB a 12GB de pendiendo el modelo de la GPU. Los programadores necesitan gestionar la transferencia de datos entre la CPU y la GPU de una manera similar a la que pasa .ºne-sided"del mensaje, una capacidad cuya ausencia en MPI ha sido considerado históricamente como una debilidad importante de MPI.

OpenMP es compatible con memoria compartida, por lo que ofrece la misma ventaja que CUDA en los esfuerzos de programación; Sin embargo, no esta diseñado para escalar más allá de un par de cientos de nodos de computación debido a a la sobrecarga en la

administración de *threads* y a los requerimientos de hardware en la jerarquia de memoria. CUDA logra mayor escalabilidad con una gestión sencilla de baja sobrecarga de *threads* y no hay requisitos de coherencia de hardware de caché. Sin embargo, CUDA no soporta la amplia gama de aplicaciones que soporta OpenMP, debido a estas ventajas y desventajas de escalabilidad. Por otra parte, muchas aplicaciones de gran escala encajan bien en el modelo simple de gestión de *threads* de CUDA y así disfrutar de la escalabilidad y rendimiento. Algunos aspectos de CUDA son similares a MPI y OpenMP, pues el programador gestiona las construcciones de código paralelos, aunque los compilaladores OpenMP hacen más de la automatización en la gestión de la ejecución en paralelo. Varios esfuerzos de investigación en curso apuntan a la adición de una mayor gestión de la automatización en paralelo y la optimización del rendimiento de la cadena de herramientas como CUDA. [10]

Estrategias de Explotación Diámica del Paralelismo

Diseñar un sistema para balancear la carga computacional en maquinas con varios núcleos heterogéneos, con diferentes aceleradores y procesadores en tiempo de ejecución, es una trabajo altamente demandante. Tradicionalmente la memoria compartida en arquitecturas homogeneas de múltiples núcleos se programan ya sea utilizando lenguajes de alto nivel (E.g.OpenMP [1], Cilk [11]) o bibliotecas de gestión de tareas de bajo nivel (por ejemplo pthread, libspe). En ambos métodos, el sistema de ejecución subyacente es el que más se dedica a la tarea de programación. Por el contrario, las máquinas heterogéneas tienen mucho más requisitos de tiempo de ejecución, ya que no proporcionan una coherente (ni siquiera compartida) memoria global. A diferencia de las CPU regulares que de forma transparente se puede acceder a toda la gama global de direcciones de memoria, aceleradores frecuentemente contienen una memoria local en la que se almacenan los datos para realizar el procesamiento. Sin ayuda del sistema de tiempo de ejecución, los programadores tendrían que explícitamente hacer cumplir la consistencia de memoria entre los aceleradores, lo que afectaría seriamente tanto la programabilidad y la portabilidad, ya que potencialmente tienen que hacer frente a diversos tipos de aceleradores (SPU de, por ejemplo, la célula con GPGPUs).

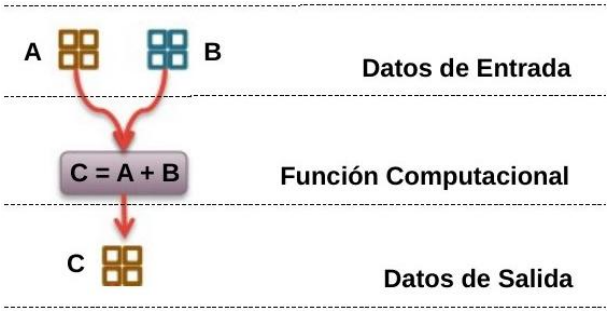
Una forma común de usar paralelismo en el desarrollo de aplicaciones es el paralelismo Maestro-Esclavo. Donde un procesador es responsable de distribuir los datos y recolectar los resultados, todos los otros procesadores ejecutan la misma tarea sobre su porción de datos. Paralelismo de datos: cada procesador ejecuta la misma tarea sobre diferentes conjuntos o subregiones de datos. Paralelismo de tareas: cada procesador ejecuta una diferente tarea.

Frameworks para la Programación Basado en Tareas

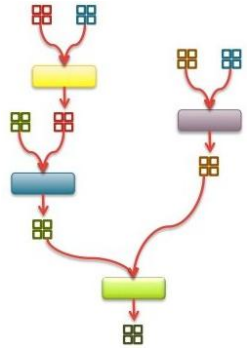
Una gestión de datos de topología jerárquica es el principal componente de nuestro sistema de tiempo de ejecución, es una instalación de gestión de datos que aspira a propor-

Figura 2.5: Modelo de Programación Basado en Tareas

(a) Componentes de una Tarea Paralela

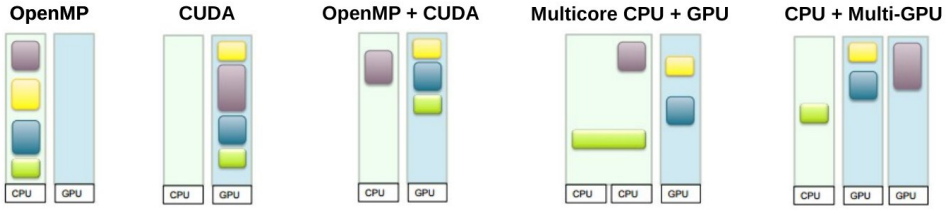


(b) Grafo Acíclico Dirigido



Fuente **Olivier Aumage.** INRIA – University of Bordeaux. March 2013

Figura 2.6: Estrategias de programación paralela para Mapear las Dependencias de Tareas Sobre los Recursos Computacionales



Fuente **Olivier Aumage.** INRIA – University of Bordeaux. March 2013

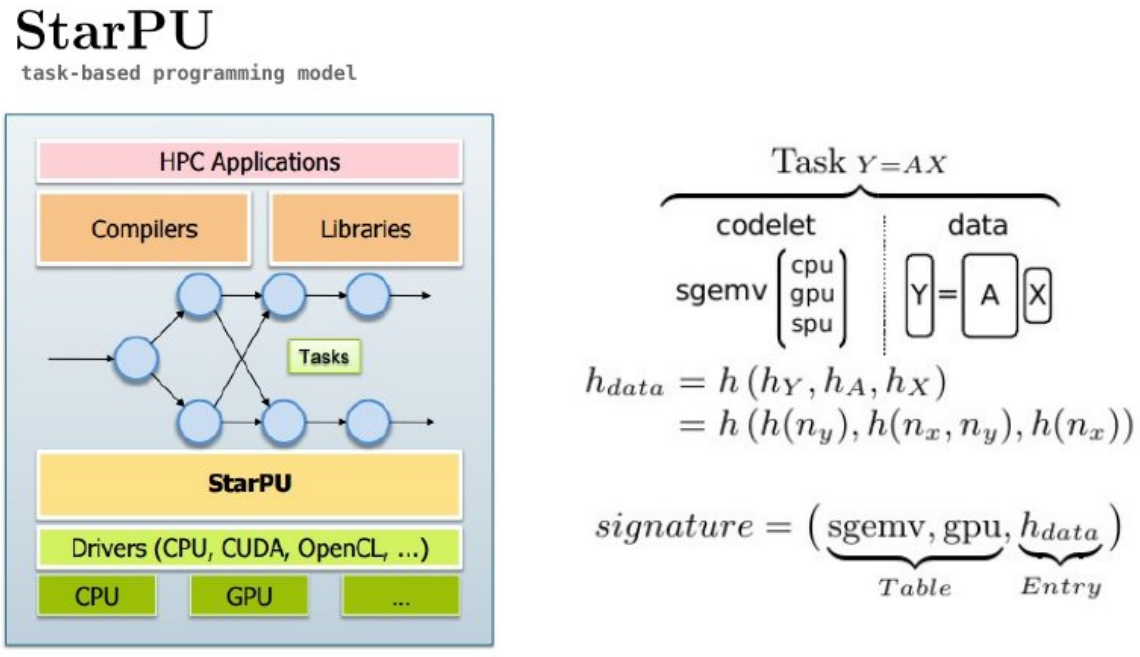
cionar una API de alto nivel que oculta la complejidad planteada por la heterogeneidad aceleradores. La idea es resumir la descripción de las regiones de memoria de manera que las transferencias de datos subyacentes y políticas de almacenamiento en caché de datos. La idea es abstraer la descripción de las regiones de memoria de manera que las transferencias de datos y almacenamiento en *data caching policies* subyacentes que se pueden optimizar, dependiendo de las capacidades del hardware.

Dadas las restricciones mencionadas impuestas por algunos aceleradores de hardware sobre los accesos a memoria, nuestro modelo de ejecución unificada se basa en el uso de codelets, que son tareas enriquecidas con una descripción de sus datos de entrada y de salida. Para ejecutar una *codelet*, cada acelerador elegible se le da una función específica para su arquitectura. Los correspondientes "kernels"son escritos por el programador,pero esperamos compilar entornos capaces de generar binarios de forma automática a partir de un código fuente genérico.

2.4. Framework de Programación Basado en Tareas: StarPU©

El modelo de trabajo adoptado por StarPU es lo suficientemente potente como para adaptarse a las diferentes tecnologías (co) procesadores. Mientras que el núcleo de StarPU se encarga de tratar los movimientos de datos a lo largo de la programación de tareas independientemente del hardware subyacente, el engrane a una nueva arquitectura sólo requiere poco esfuerzo. un controlador de este tipo debe primero proporcionar las funciones de memoria para transferir entre el *host* y el acelerador (ver Figura 2.7). A continuación, debe implementar el método que ejecuta realmente (o descargar) una tarea en los aceleradores, por lo general por los medios de interfaz propietaria de la arquitectura o con alguna biblioteca específica arquitectura de terceros. Este modelo se aplicó con éxito en la parte superior de los procesadores multi-núcleo, en las GPU con CUDA, y en el procesador Cell [11].

Figura 2.7: StarPU como framework de Programación bada en tareas



2.4.1. Planificador Interno de Tareas de StarPU©

Las transferencias de datos tienen un impacto importante en el rendimiento, por lo que favorece una planificación local, lo que puede aumentar los beneficios de las técnicas de almacenamiento en caché mediante la mejora de la reutilización de datos. Teniendo en cuenta que múltiples problemas se pueden resolver al mismo tiempo, y que las máquinas no están totalmente dedicadas (por ejemplo, cuando el acoplamiento de códigos), se puede

considerar que la programación dinámica se convierte en una necesidad. En el contexto de plataformas heterogéneas, el rendimiento varía mucho de acuerdo con arquitecturas y de acuerdo con la carga de trabajo (por ejemplo, código SIMD vs. acceso a la memoria irregular). Por lo tanto,es crucial tener en cuenta la especificidad de cada unidad de cálculo al asignar el trabajo.

2.4.2. Balanceo de Cargas y Regulación de Frecuencias

Soluciones fundamentadas para ahorrar el consumo de energía de la CPU.
Las técnicas generales de ahorro de energía de la CPU se han estado investigado extensamente y por tanto en la actualidad existe un estándar para la administración del consumo energético a través del sistema de escalado dinámico de voltaje DVFS, donde el voltaje utilizado en un componente se aumen- ta o disminuye, dependiendo de las circunstancias, el escalado dinámico de voltaje para reducir el voltaje se conoce como undervolting; y el escalado dinámico de voltaje para aumentar el voltaje se conoce como overvolting.

En particular, algunos proyectos han tratado de mejorar la eficiencia energética de los multiprocesadores ejecutando aplicaciones paralelas. Por ejemplo Li et al. (11) Propone una solución llamada barreras ahorrativas 'The thrifty barrier', donde asigna los núcleos más rápidos en un modo de potencia inferior a las barreras (es decir, puntos de encuentro) a la espera de los núcleos más lentos por lo que se puede ahorrar energía.

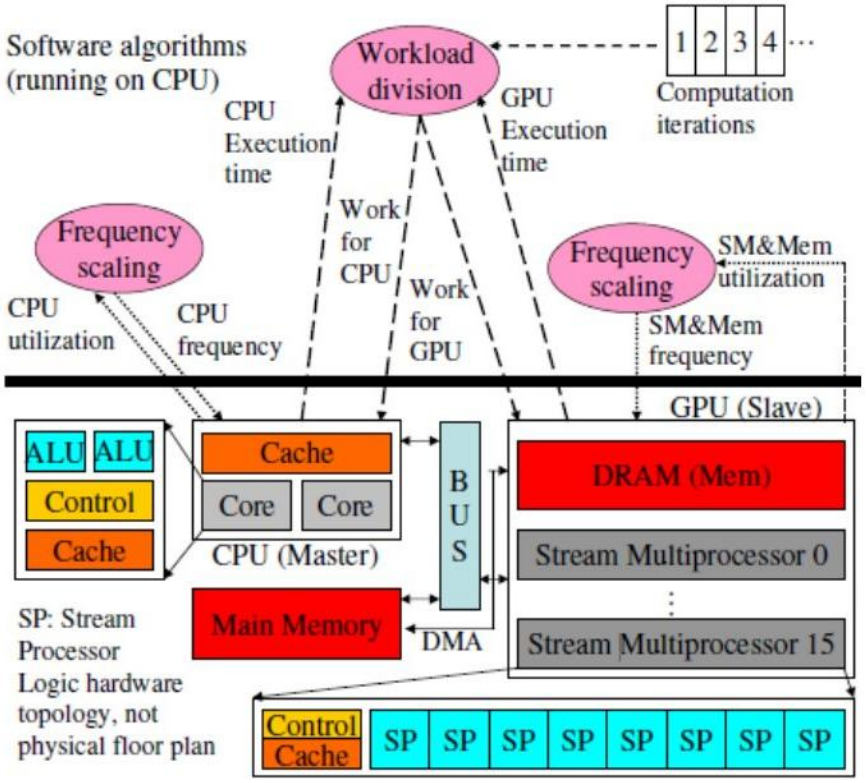
Liu et al. en [12], utilizan por núcleo DVFS, para reducir la frecuencias de los *cores* más rápidos, de tal manera que tanto el tiempo de inactividad debido a la espera y el consumo de energía se reduce. Sin embargo, estos estudios se centran en las arquitecturas de CPU sin considerar la GPU como parte del sistema, por lo que sus métodos no se pueden aplicar directamente a las arquitecturas heterogéneas tipo CPU-GPU.

Soluciones fundamentadas para establecer la eficiencia energética coordinando los núcleos de la GPUs o memoria.
Lee et al. en [13], han experimentado como coordinar el número de núcleos activos, la frecuencia de la parte central y parte de la cache en la GPU. Sin embargo, la coordinación a nivel de finalización de iteraciones de la carga de trabajo entre CPU y GPU no se expresa en su trabajo.

GreenGPU es un framework de balanceo y regularización de frencuencía holístico para la Eficiencia Energética Sobre Arquiteturas Heterogéneas GPU-CPU [14]. GreenGPU en un primer nivel regula dinámicamente las frecuencias de los núcleos de la GPU y la memoria de forma coordinada, en base al porcentaje de utilización, para maximizar el ahorro de energía, con una mínima degradación del rendimiento marginal. Del mismo modo la frecuencia y el voltaje de la CPU se regulan de manera similar a la de la GPU, ya que este framework adoptó la técnica para la administración del consumo energético a través del sistema de escalado dinámico de voltaje DVFS.

En el segundo nivel del framework GreenGPU divide de forma dinámica y distribuye las cargas de trabajo entre la GPU y CPU, basados en el porcentaje de trabajos a resolver sincronizando los tiempos de finalización de procesamiento por iteración, de tal manera que ambas partes puedan finalizar aproximadamente al mismo tiempo. Como resultado se ahorra energía desperdiciada cuando la CPU o la GPU se encuentran en marcha muerta.

Figura 2.8: GreenGPU framework de balanceo y regularización de frencuencía



Fuente **Kai Ma et al.** GreenGPU – IEEE Computer Society, 2012.

Capítulo 3

Métricas y Aplicación de Caso de Estudio: Ondes3D©

3.1. Metricas de Consumo de Energía

Múltiples capturas de potencia a lo largo del tiempo permiten determinar el consumo de energía en un sistema. Por lo tanto, la cantidad de energía utilizada en la ejecución de algoritmos para obtener la solución, es llamada *energy-to-solution* por Padoin et al. en [15]. En este camino, es necesario integrar la potencia instantánea a través del tiempo, para estimar la energía consumida en la ejecución de aplicaciones, como se muestra en la Ecuación 3.1. En la practica se utiliza la discretización del consumo de potencia por unidad de tiempo, como equivalente de usar la integral de energía (ver Ecuación 3.2). Por lo tanto, cada muestra i es un instante de potencia en un intervalo de tiempo Δt, para representar el consumo de energía mediante muestras de potencia a través del tiempo (ver Figura 3.1). [16]

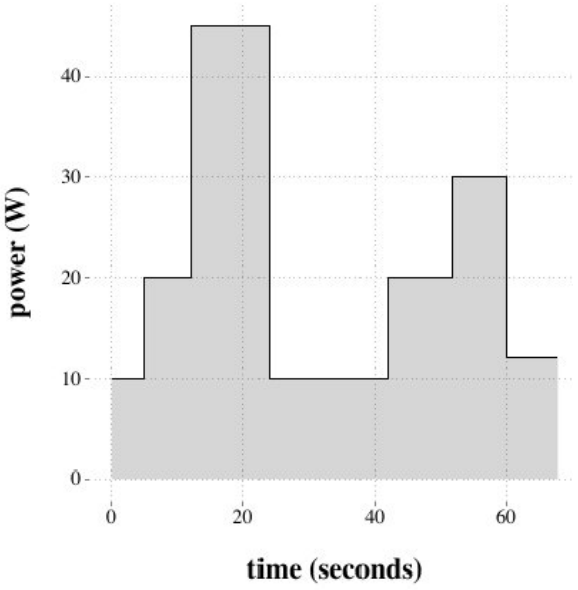
$$Energy = \int Power(t)\ dt \tag{3.1}$$

$$Energy = \sum_{i=1}^{N} Power(i) * \Delta t(i) \tag{3.2}$$

Este trabajo propone la ecuación 3.3 para calcular la energía consumida en la ejecución de aplicaciones en los nodos del *cluster* GUANE, denotado como $E_{Solution}$. Esta ecuación utiliza la discretización del consumo de potencia por unidad de tiempo, para cada muestra i como una potencia instantánea por nodo, denotado $Power_{Node}(i)$ en un intervalo de tiempo Δt. El tiempo total de ejecución está dividido por el intervalo de tiempo usado por un grupo de CPU y un grupo de GPU denotado Δt_{CPU} y Δt_{GPU} respectivamente, para procesar un porcentaje de la carga de trabajo de una misma aplicación. Siendo N el tiempo máximo que tarda un grupo de CPU o GPU en procesar una aplicación.

$$E_{Solution} = \sum_{i=1}^{N} Power_{Node}(i) * max\ (\Delta t_{CPU}(i), \Delta t_{GPU}(i)) \tag{3.3}$$

Figura 3.1: Discretización del Consumo de Potencia por Unidad de Tiempo.



Fuente **Gustavo Rostirolla.** GreenHPC: A Novel Framework to Measure Energy Consumption on HPC Applications. 2015

El consumo instantáneo de potencia es el volumen de energía entregado en un instante de tiempo especifico [16]. Este trabajo estima el consumo de potencia instantáneo por nodo $Power_{Node}$, como la suma de cada medida de consumo de potencia por componente (CPU, GPU y RAM), como se representa en la ecuación 3.4. Donde P^j_{CPU} es el consumo de potencia total por cada procesador homogéneo en un mismo nodo y nc es el total de procesadores. P^j_{GPU} es el consumo de potencia total por cada acelerador homogéneo en un mismo nodo y ng es el total de aceleradores. P^j_{RAM} es el consumo de potencia total por cada memoria principal en un mismo nodo y nm es el total de memorias.

$$Power_{Node}(i) = \sum_{j=1}^{nc} P^j_{CPU}(i) + \sum_{j=1}^{ng} P^j_{GPU}(i) + \sum_{j=1}^{nm} P^j_{RAM}(i) \tag{3.4}$$

3.1.1. Ejemplo de Uso de la Ecuación $E_{Solution}$

Se desea calcular la energía consumida en la ejecución de un subprograma BLAS3, el cual resuelve una multiplicación de matrices, de la forma: $GEMM : C = \alpha * C + \beta * A * B$. La cual se procesará utilizando una CPU (Intel® Xeon® E5645, $2.40 GHz$) de la configuración 'A' de los nodos del *cluster* GUANE. Efectivamente, el tiempo máximo está determinado por el tiempo que tarda la CPU en finalizar la ejecución $\Delta t_{CPU}(i)$, en cambio $\Delta t_{GPU}(i) = 0$ porque no se utilizarán GPU para procesamiento en este ejemplo. Sin embargo, para calcular el total de la potencia consumida por nodo $Power_{Node}$, se debe tener en cuenta la suma de potencia de cada uno de los componentes, debido a que estos

se encuentran encendidos y conectados a la misma fuente de poder dentro del nodo.

Para calcular el tiempo total de ejecución en la CPU $\Delta t_{CPU}(N)$ se asume solapamiento entre: el tiempo para transferir los datos, desde la memoria principal hasta la memoria *cache* de la CPU y el tiempo para realizar el procesamiento en la CPU, denotado como $time_{comm}$ y $time_{comp}$ respectivamente. Este solapamiento es expresado en la Ecuación 3.5 por Jack Dongarra en [17].

$$\Delta t_{CPU}(N) = max\ (time_{comm}, time_{comp}) \tag{3.5}$$

El algoritmo BLAS3 tiene una complejidad de $2n^3 FLOP$ y $3n^2$, para realizar el procesamiento y las transferencias en memoria, respectivamente. Si la matriz A y B tienen una dimensión de $n = 700$, el tamaño de los datos es $3{,}92 MBytes$ por matriz. Por lo tanto, el tiempo para mover los datos hasta la memoria *cache* es de $0{,}367ms$ con $32GB/s$ de ancho de banda; el tiempo para realizar el procesamiento es $11{,}9ms$ con $57{,}6GFLOP/s$ de *Rpeak*. Como resultado el tiempo total de ejecución del subprograma BLAS3 es $\Delta t_{CPU}(N) = 11{,}9ms$.

El consumo de potencia medio por CPU es $P^j_{CPU} = 80watts$, cuando el procesador esta operando a una frecuencia base; el consumo de potencia medio por GPU es $P^j_{GPU} = 87{,}21watts$ cuando el acelerador esta operando a una frecuencia base; y el consumo de potencia medio por RAM es $P^j_{RAM} = 70watts$, estos estados son llamados *baseline*.

Finalmente, la energía requerida para ejecutar el subprograma BLAS3 es $E_{Solution} = 11{,}186Kilojoules$ con una potencia total $Power_{Node} = 940watts$ y un tiempo de ejecución $\Delta t_{CPU}(N) = 11{,}9ms$. No obstante, este resultado tiene baja eficiencia energetica si es comparado con el consumo de energía deseado, donde se contabiliza el consumo de potencia por la cantidad de componentes utilizados para procesamiento. El consumo de energía deseado es $0{,}95Kilojoules$ con un consumo de potencia total $Power_{Node} = 80watts$ por la CPU utilizada en el procesamiento. Este trabajo estudia la técnica de regulación de frecuencia para administrar el consumo de potencia por cada GPU *(Power Capping)*, habilitando solo el componente que está siendo utilizado para procesamiento y disminuyendo las frecuencias de los componentes que no son utilizados. Ver Capítulo 4.

3.2. Aplicación de Caso de Estudio: Ondes3D©

3.2.1. Ecuación Elastodinámica

La simulación de la propagación de ondas sísmicas: es una herramienta necesaria en geofísica para un eficiente análisis de fuertes movimientos y mitigación de riesgos. Por lo tanto, la evaluación de los daños que pueda producirse durante el fuerte movimiento del suelo, es crítico para la planeación de zonas urbanas, como lo describe Victor Martínez et al. en [18]. Esta clase de problemas pueden ser descritos por la ecuación de onda sísmica:

$$\frac{\partial v_i}{\partial t} = \frac{\partial \sigma_{ij}}{\partial j} + F_i \tag{3.6}$$

Además, se considera la ecuación de onda sísmica para un medio solido, por lo tanto, se presenta la relación constitutiva en el caso de un medio isótropo, descrita por Victor Martínez et al. en [19] como:

$$\frac{\partial \sigma_{ij}}{\partial t} = \lambda \delta_{ij}(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}) + \mu(\frac{\partial v_i}{\partial j} + \frac{\partial v_j}{\partial i}) \tag{3.7}$$

Los índices *i, j, k* representan un componente del vector o un campo de tensores en las coordenadas (*x, y, z*), v_i y σ_{ij} representan la velocidad y el campo de esfuerzo respectivamente, y F_i denota una fuente de fuerza externa. ρ es la densidad del material, λ y μ son los coeficientes elasticos conocidos como parametros de Lamé. La derivada del tiempo esta denotado por $\frac{\partial}{\partial t}$ y la derivada espacial con respecto a la dirección i-th denotado por $\frac{\partial}{\partial i}$. El simbolo de Kronecker δ_{ij} es igual a 1 si i=j y 0 en caso contrario. Los exponentes *i, j, k* indican la dirección espacial con dirección $\sigma^{ijk} = \sigma(i\Delta s, j\Delta s, k\Delta s)$, Δs corresponde al paso en el espacio y Δt al paso en el tiempo. Por consiguiente, se utiliza el método de diferencias finitas estándar para resolver la ecuación elastodinámica. Este es un enfoque presentado por Jean Virieux en [20], el cual sigue siendo ampliamente utilizado debido a su simplicidad de ejecución para una amplia gama de aplicaciones. Teniendo en cuenta el cuarto grado clásico en el espacio y segundo grado en la aproximación de tiempo, la Ecuación 3.8 expresa la solución de la velocidad mediante diferencias finitas. Se utiliza el mismo esquema numérico para solucionar el campo de esfuerzo.

$$\begin{aligned} v_x^{(i+\frac{1}{2})jk}(l+\frac{1}{2}) = v_x^{(i+\frac{1}{2})jk}(l-\frac{1}{2}) + a_1 F_x^{(i+\frac{1}{2})jk} \\ + a_2[\frac{\sigma_{xx}^{(i+1)jk} - \sigma_{xx}^{ijk}}{\Delta_x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k} - \sigma_{xy}^{(i+\frac{1}{2})(j-\frac{1}{2})k}}{\Delta_y} \\ + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})} - \sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})}}{\Delta_z}] \\ - a_3[\frac{\sigma_{xx}^{(i+2)jk} - \sigma_{xx}^{(i-1)jk}}{\Delta_x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{3}{2})k} - \sigma_{xy}^{(i+\frac{1}{2})(j-\frac{3}{2})k}}{\Delta_y} \\ + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{3}{2})} - \sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{3}{2})}}{\Delta_z}] \end{aligned} \tag{3.8}$$

3.2.2. Estructura y Ejecución de Ondes3D© sobre StarPU©

Las ecuaciones anteriores se encuentran resueltas en el paquete de software Ondes3D©, desarrollado por BRGM[1]. Ondes3D© resuelve el método de diferencias finitas a través de

[1]BRGM (*Bureau de Recherches Géologiques et Minières:* Oficina de Investigación Geológica y Minera), es la institución de referencia pública de Francia para aplicaciones de ciencias de la tierra en el área de gestión de recursos, el subsuelo y sus riesgos. ver http://www.brgm.eu/

mallas estructuradas *Structured Grid*, para subdividir el área de la simulación de propagación de ondas sísmicas en bloques de datos de entrada, y formular códigos *stencil* mediante *kernels* computacionales para explotar el paralelismo de datos. Mientras, StarPU© es un *framework* de programación basado en tareas, que trabaja con diferentes niveles para integrar: librerías de HPC y variables de entorno con los *kernels* computacionales; proporcionando una interfaz que unifica la ejecución sobre diferentes procesadores y aceleradores; como resultado, aplicaciones que aprovechan eficientemente diferentes recursos computacionales. De acuerdo a la descripción realizada por Cédric Augonnet et al. en [**?**].

Para escribir una aplicación mediante codigos *stencil* en CUDA©, los programadores deben concentrarce en: la asignación de memoria explícita en la GPU, la copia de datos desde la memoria principal a la memoria de la GPU, especificar el tamaño del bloque y el grid, escribir el *kernel* del programa, y llamar el *kernel*. Asi mismo el programador debe distribuir los datos y los cálculos a través de los *threads* en CUDA©. [21]

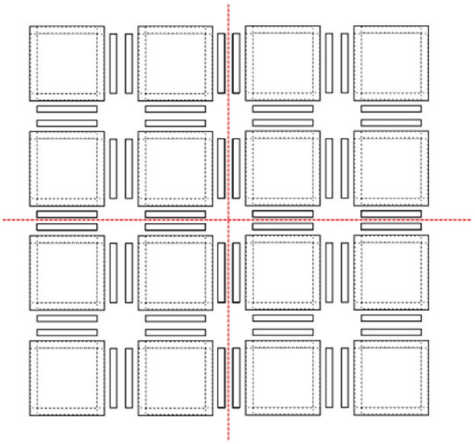
Los codigos *stencil* son utilizados para calcular los componentes de velocidad y el campo de esfuerzo. Para expresar el paralelismo de tareas en Ondes3D©, la clave es dividir el modelo sismico en una malla de bloques de grano fino, cada parte del código se ejecuta en un solo bloque y se convierte en una tarea (ver Figura 3.2a). Por lo tanto, la ejecución del sistema permite descomponer el problema original en varias tareas *(computation task or communications task)*, creando un Grafo Acíclico Dirigido (DAG) que será distribuido sobre los recursos computacionales heterogéneos de la misma maquina (ver Figura 3.2b). Este esquema es ampliamente utilizado en los ejemplos guía de StarPU©, de hecho, uno de ellos se llama *stencil* e implementa el juego de la vida para demostrar cómo utilizar StarPU© con códigos *stencil*.

Ondes3D© es la aplicación seleccionada como caso de uso para realizar la integración con el *framework* StarPU©. Asi como, para analizar la eficiencia energética y la portabilidad de aplicaciones sobre diferentes arquitecturas heterogéneas. El Algoritmo 1, expresa las actividades procedimentales requeridas en StarPU© para portar Ondes3D©, donde el desarrollador: primero debe realizar el registro y la liberación de datos (vectores, matrices y entre otros); seguidamente realiza la definición del *codelet* (el *kernel* para la CPU y la GPU); y por ultimo realiza la definición y el envío de la Tarea. Una descripción detallada de estas implementaciones se puede encontrar en [22, 23].

El Algoritmo 1. descompone el problema en dos tipos de tareas para procesar los bloques *stencil* en cada paso de tiempo, de forma iterativa: las tareas de comunicación actualizan los datos en el código (guarda y registra las condiciones de frontera), para la sincronización de *threads* y compartir datos entre los *cores* heterogéneos; las tareas de computación resuelven el método de diferencias finitas, descrito en la Ecuación 3.8, para hallar la velocidad y el esfuerzo en cada bloque *stencil*.

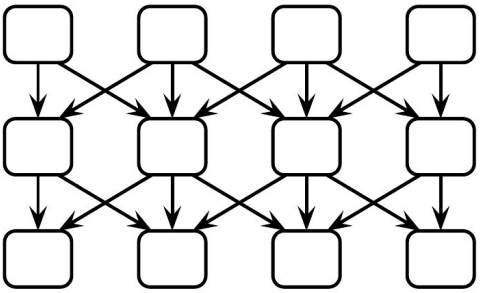
Figura 3.2: Mapeo de Tareas de Ondes3D© sobre StarPU©

(a) Modelo Sísmico 2D Dividido en Bloques Stencil



Fuente **Victor M. et al.** Towards Seismic Wave Modeling on Heterogeneous Manycore Architectures Using Task-Based Runtime System.

(b) Representación del Grafo de Dependencias de Tareas para Códigos Tipo Stencil



Fuente **Andrei Radulescu et al.** Fast and Effective Task Scheduling in Heterogeneous Systems.

3.3. Análisis de Eficiencia Energética y Portabilidad

3.3.1. Diseño Experimental para Ondes3D© sobre StarPU©

El objetivo de este análisis es demostrar la portabilidad de aplicaciones en tres arquitecturas heterogéneas: un nodo de la configuración 'A' del *cluster* GUANE, el servidor SALMAO y especialmente en dispositivos embebidos como la NVIDIA® Jetson TK1, la cual representa una alternativa de bajo consumo de energía, para la simulación de la propagación de ondas sísmicas. Para realizar una comparación ecuánime sobre las diferentes arquitecturas heterogéneas, se plantea el mismo modelo sísmico tridimensional con una área de tierra de $(300km * 300km * 80km)$ que genera una malla cartesiana de 7,2 millones de puntos y $625MB$ de espacio en memoria principal. En cuanto a la implementación de Ondes3D© en StarPU©, se debe de programar dos *kernels* para calcular la velocidad y el campo de esfuerzo en la CPU y la GPU. Resultando 36 tareas paralelas de computación (20 para la velocidad y 16 para el esfuerzo) y 144 tareas paralelas de comunicación en intercambio de datos entre los bloques *stencil*.

El sistema de ejecución de StarPU© utiliza dos banderas para asignar los *cores* heterogéneos que están disponibles para la ejecución de la simulación, llamados *worker*: la bandera *STARPU_NCPU* define el número de *cores* de la CPU que serán usados en el procesamiento; y la bandera *STARPU_NCUDA* define el numero de GPU utilizadas. En este escenario experimental se utiliza $3 cores$ para procesamiento y $1 core$ para realizar tareas

Algorithm 1 Algoritmo para mapeo de tareas de Ondes3D en StarPU

```
1: procedure Initialization of data structures
2:     Read_parameters(file)
3:     Model_slicing(Create blocks)
4: procedure Create all tasks
5:     Create dummy start task(Time measurement)
6:     for each iteration do
7:         for each block do
8:             Create task(update source)
9:             Create task(compute velocity)
10:            Create task(save boundaries for velocity)
11:            Create task(compute strees)
12:            Create task(save boundaries for stress)
13:            Create task(record Syncronization)
14:        Create dummy end task(synchronization)
15:    Release dummy start task
16:    Wait(end of the dummy end task)
```

de administración de la GPU. También se utiliza la bandera *STARPU_CUDA_ASYNC* para ejecutar los CUDA© *cores* de manera concurrente, si está disponible en la GPU.

El planificador de tareas seleccionado en StarPU© es DMDAR *(Deque Model Data Aware Ready)* para asignar tareas a los recursos computacionales *workers* buscando el mejor rendimiento computacional basado en la historia. De tal manera que las colas de tareas son ordenadas según el número de *workers* disponibles. Para hacer este experimento, se trabajó con la versión 1.2 de StarPU©. Las métricas utilizadas para realizar el análisis de eficiencia energética y portabilidad son: el tiempo para solución (tiempo requerido para finalizar la ejecución por cada experimento); el consumo de energía, el cual se calcula usando la herramienta de NVIDIA-SMI para capturar la potencia en *Watts* y calcular la energía en *Joules*.

3.3.2. Resultados de Ondes3D© sobre StarPU©

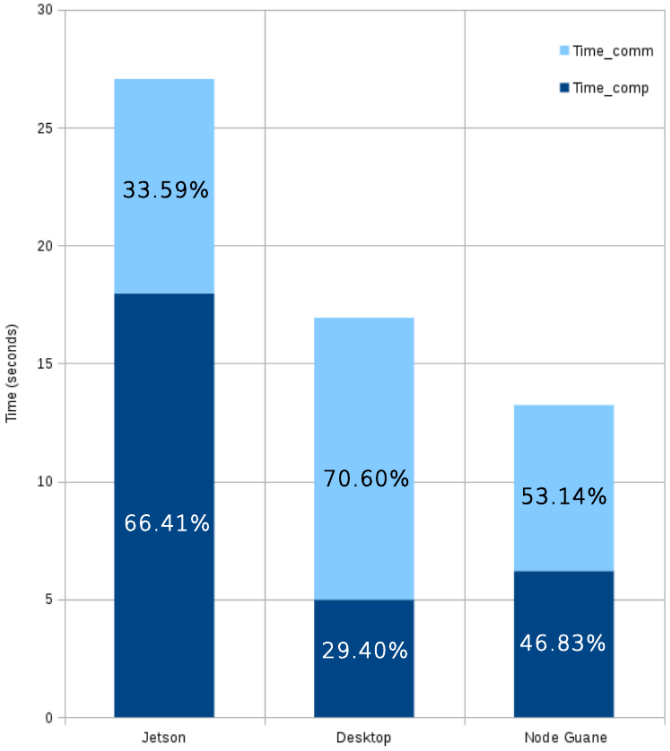
Los resultados descritos en la Figura 3.3 confirman la asunción de que la plataforma embebida Jetson TK1 con (27.06 segundos) exhibe un tiempo de solución menor en comparación con el nodo GUANE y el servidor SALMAO. Resultando que el nodo GUANE con (13.23 segundos) es 2x más rápido que la plataforma embebida Jetson TK1 y la relación con el servidor SALMAO es de 1.59x con (16.94 segundos) en tiempo de solución. Dado que la tarea de experimentación para Ondes3D fue una malla cartesiana de 7,2 millones de puntos que representa $625MB$ de espacio en memoria principal, donde la carga de trabajo se divide en 36 *Tareas de Procesamiento* para resolver la velocidad y el esfuer-

zo. Al mismo tiempo se tiene 144 *Tareas de Comunicación* para compartir los datos entre cada bloque stencil de la cuadricula primaria, presentando una constante comunicación entre la memoria RAM y la memoria global de la GPU, tanto para transferencia de datos como para sincronización de subprocesos.

Debido a que el tamaño de la tarea de experimentación (625 MB) fue ajustado al espacio disponible de la memoria RAM de la plataforma embebida Jetson TK1, StarPU© utiliza los 3 núcleos de la CPU y un *Streaming Multiprocessor* de GPU para procesar las tareas de cúmputo. El sistema de ejecución de StarPU© sólo planifica las tareas de cómputo en la GPU y las tareas de comunicación son administradas por los núcleos de la CPU, para el servidor SALMAO y el nodo GUANE.

Por consiguiente se puede observar en la Figura 3.3 que la plataforma embebida Jetson TK1 utilza el 66.41 % del tiempo de solución en tareas de procesamiento, mientras que el servidor SALMAO y el nodo GUANE solamente utilizan el 29.40 % y 46.83 % del tiempo de solución respectivamente. Esto se debe a que la plataforma embebida Jetson TK1 tiene una memoria compartida entre los núcleos de CPU y GPU, disminuyendo el tiempo de transferencia de datos entre la memoria RAM y la memoria global de la GPU para procesar las 144 *Tareas de Comunicación* y como resultado, sólo se cuenta el tiempo de sincronización de los subprocesos de la GPU y las funciones de la CPU.

Figura 3.3: Comparación del tiempo transcurrido hasta la solución entre la NVIDIA Jetson, el nodo GUANE y el servidor SALMAO.



Para realizar el análisis de eficiencia energetica se estableció un consumo de potencia medio de (15 Watts), para el procesador hibrido NVIDIA® TEGRA® K1^{TM} tomado desde las especificaciones establecidas para la plataforma embebida Jetson TK1. Donde (10 Watts) están determinados por la utilización de la GPU y los (5 Watts) restantes por la CPU. Para medir el consumo de potencia del servidor SALMAO y del nodo GUANE se utilizó la interfaz *nvidia-smi*, donde se observó un consumo de potencia medio de (135.6 Watts) en la GPU para el servidor SALMAO, con un limite superior de (191.35 Watts) para tareas de procesamiento y un nivel medio de (80 Watts) para el procesamiento. En cambio el servidor GUANE presentó un consumo de potencia medio de (123.4 Watts).

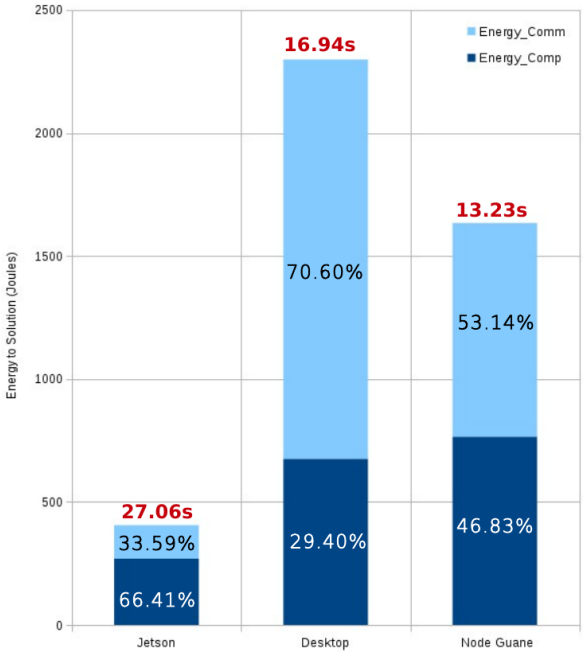
Al comparar las tres plataformas desde la perspectiva de energía empleada por la GPU para realizar la tarea de experimentación, se puede observar en la Figura 3.4a que la plataforma embebida Jetson TK1 realizó la tarea de manera más enérgeticamente eficiente que las otras dos plataformas. Donde la Jetson TK1 utilizó (405.9 Julios), mientras que el servidor SALMAO y el nodo GUANE utilizaron (2298.3 Julios) y (1633.7 Julios) respectivamente.

Dado que el procesador NVIDIA® TEGRA® K1^{TM} tiene incluido el consumo de energía empleada por la GPU y la CPU. Por con siguiente se debe incluir en la comparación el consumo energetico por utilización de la CPU: resultando el servidor SALMAO con (4500.5 Julios) y el nodo GUANE con (2692.1 Julios) de consumo de energía empleada por la GPU y la CPU, como se observa en la Figura 3.4b .

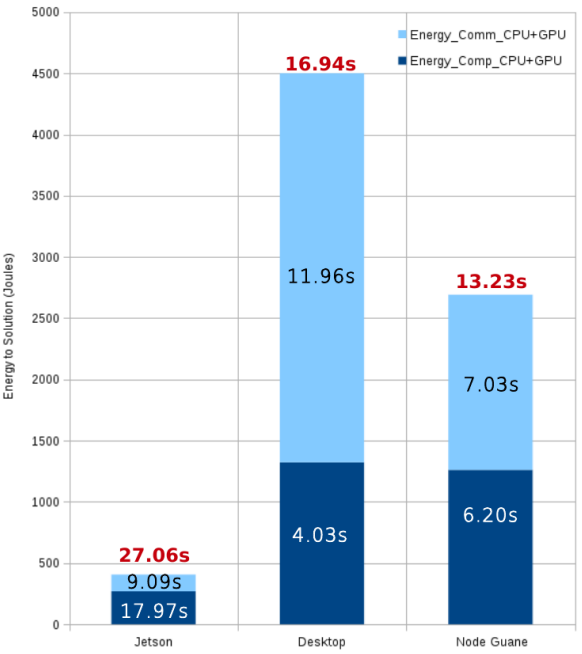
La Figura 3.4c muestra la suma del consumo de energía utilizado por los recursos computacionales que se encontraban inactivos en el nodo GUANE al momento de procesar la tarea de experimentación, se puede concluir que el nodo GUANE con (5470 Joules) de energía empleada es la plataforma menos eficiente enérgeticamente para procesar tamaños de tarea de esta magnitud, dado que el 50 % del consumo de energía es aportado por los recursos computacionales inactivos en el nodo GUANE. Siendo este un primer escenario del problema de investigación donde se debe seleccionar el número óptimo de recursos computacionales para dividir y realizar un buen mapeo de tareas dada la complejidad de la tarea y su tamaño en memoria, el cual se discutirá en los siguientes capítulos.

Figura 3.4: Ananálisis de eficiencia energetica para la ejecución de Ondes3D© sobre tres diferentes plataformas.

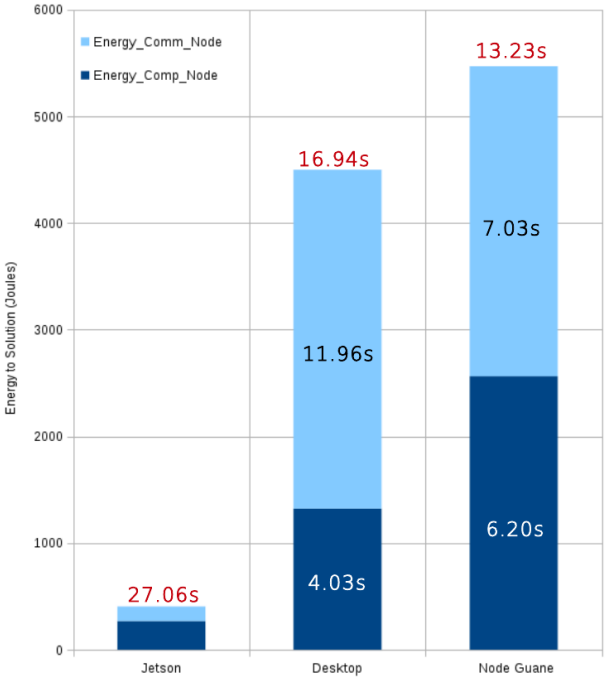
(a) Energía empleada por la GPU.



(b) Energía consumida por (CPU + GPU).



(c) Energía consumida por la plataforma.



Capítulo 4

enerGyPU para la Evaluación de Rendimiento y Consumo de Potencia

4.1. LINPACK Benchmark

El LINPACK Benchmark está conformado por un conjunto de subrutinas Fortran, que resuelve un sistema de ecuaciones lineales denso con dimensión $N * N$, a través del método de eliminación gaussiana con pivote parcial. La salida del Benchmark describe el rendimiento computacional utilizado, para resolver un problema de la forma $Ax = b$, en este caso, A es la matriz densa a descomponer. El Benchmark varía el tamaño del problema para buscar la mayor tasa de operaciones punto flotante a través del tiempo y determinar el mejor rendimiento computacional *Rmax*. La complejidad computacional para resolver este sistema de ecuaciones lineales es $2n^3/3 + 2n^2$, independiente de el método usado para buscar la solución del problema [24].

4.1.1. HPL-2.0 Optimizado para NVIDIA® Tesla GPU

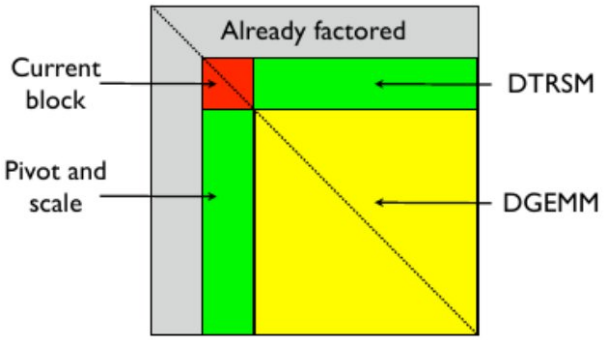
El paquete de software HPL-2.0 desarrollado por Antoine Petitet et al. en [25], está configurado para la familia de GPU Tesla 20-series. Este software es diseñado para acelerar el LINPACK Benchmark sobre *cluster* heterogéneos, donde los *cores* de la CPU y los CUDA© *cores* de la GPU son utilizadas en sinergia con mínimas modificaciones del código original. Está implementación usa las librerías de álgebra lineal (BLAS) definidas por la libreria MKL de Intel® y CUBLAS de NVIDIA®. Las cuales interceptan las llamadas al DGEMM y al DTRSM para procesar simultaneamente sobre la CPU y GPU. Una amplia descripción sobre la implementación del HPL-2.0 es realizada por Massimiliano Fatica en [26].

El algoritmo utilizado en la paralelización del HPL-2.0 se basa en la descomposición LU con pivote parcial. La Figura 4.1a indica la factorización LU: el area gris representa la porción de la matriz ya factorizada. El área roja es el bloque actual que se está factorizando. Una vez que esta factorización está listo, se aplica a la submatriz de la derecha. El

ultimo paso es actualizar la salida de la submatriz en amarillo. El algoritmo LU se puede caracterizar por la dimensión de la matriz de datos de entrada, ya que el número de nodos y vértices del Grafo DAG solo depende del tamaño de la matriz (ver Figura 4.1b). Por lo tanto, este algoritmo es linealmente paralelo a diferencia del problema sísmico, donde los códigos *estencil* tienen dependencias de datos entre los nodos del DAG.

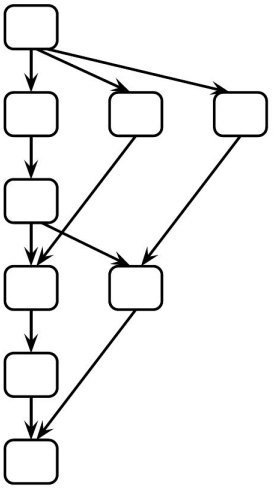
Figura 4.1: Mapeo de Tareas de DGEMM del LINPACK Benchmark

(a) Factorización LU para Resolver un sistema de ecuaciones lineales denso



Fuente **Massimiliano F. et al.** Accelerating Linpack with CUDA on heterogeneous clusters.

(b) Representación del Grafo de Dependencias de Tareas para la Factorización LU



Fuente **Andrei Radulescu et al.** Fast and Effective Task Scheduling in Heterogeneous Systems.

4.1.2. Carga de Trabajo: DGEMM

El mayor porcentaje de procesamiento se concentra en la multiplicación de matrices llamada DGEMM. Las matrices de entrada al DGEMM tienen un formato de datos en doble precisión y vienen dadas de la forma: A(M, K), B(K, N), C(M, N), para resolver el sistema de ecuaciones lineales $C = \alpha AB + \beta C$. La subrutina BLAS puede controlar las dimensiones principales de las matrices A, B and C (lda, ldb and ldc) para diferentes M, K y N, como se muestra en la Figura 3a y se puede realizar la llamada a la subrutina BLAS, de la siguiente forma:

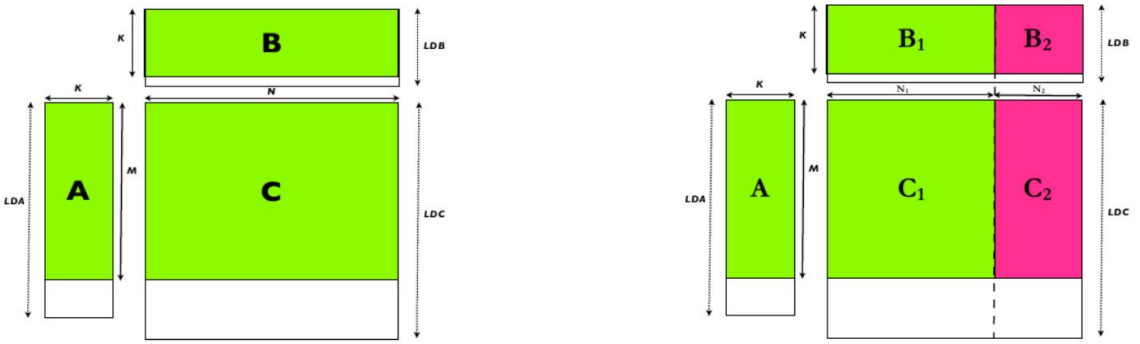
DGEMM ('N','N',m n,k,alpha,A,lda,B,ldb,beta,C,ldc)

Por consiguiente la subrutina BLAS puede ser utilizada para interceptar las llamadas al DGEMM y dividir en N_n bloques las matrices B y C, como se muestra en la Figura 3b. De manera que los N_n bloques puedan ser procesados simultaneamente sobre la CPU y GPU, la llamada a la subrutina BLAS se realiza de la siguiente forma:

DGEMM ('N','N',m,n1,k,alpha,A,lda,B1,ldb,beta,C1,ldc)

DGEMM ('N','N',m,n2,k,alpha,A,lda,B2,ldb,beta,C2,ldc)

Figura 4.2: Carga de Trabajo - DGEMM Multiplicación Matriz-Matriz.



(a) El Area Verde representa la carga de trabajo DGEMM que puede ser ejecutado por un recurso computacional como la CPU o la GPU.

(b) El Area Verde es la porción de datos DGEMM que puede ser ejecutado sobre la GPU en sinergía con el Area Rosa DGEMM que puede ser ejecutado sobre la CPU.

Fuente **Massimiliano F. et al.** Accelerating Linpack with CUDA on heterogeneous clusters.

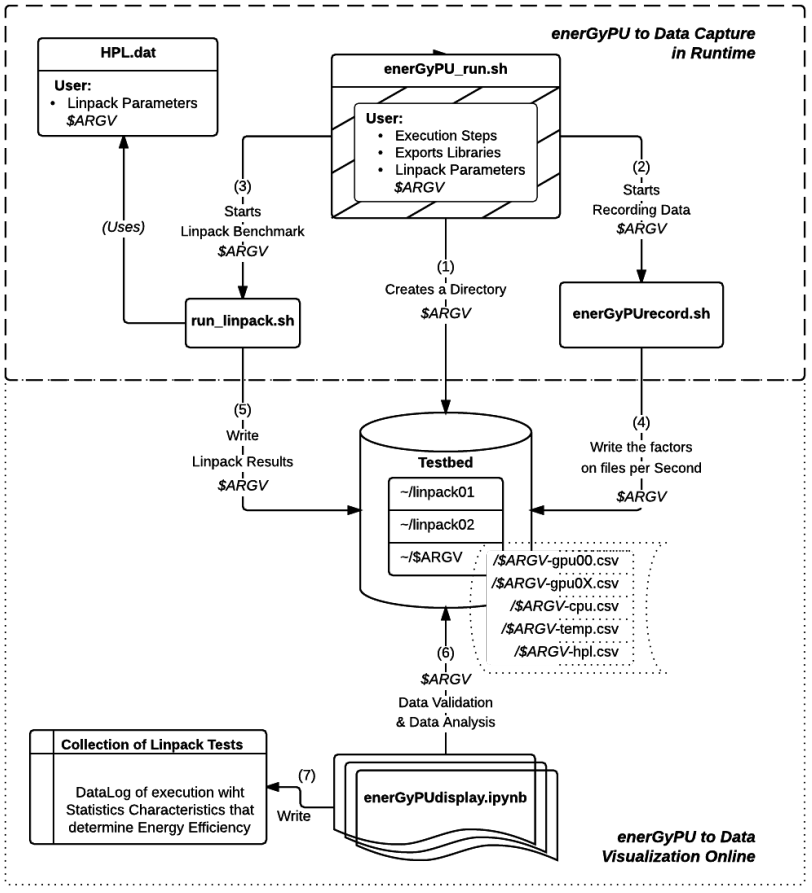
4.2. enerGyPU Monitor de Rendimiento y Consumo de Potencia

enerGyPU es una herramienta de monitorización creada para analizar múltiples ejecuciones en aplicaciones de gran escala, combinando diferentes parámetros del problema y de la arquitectura, para controlar los factores computacionales que determinan la eficiencia energética en términos de *energy-to-solution* sobre *clusters* heterogéneos con múltiples GPU. enerGyPU centraliza y automatiza la captura de datos en tiempo de ejecución, mientras se ejecuta en paralelo con las aplicaciones. Adicionalmente despliega información a través de gráficos de linea de tiempo, gráficos de barras, tablas estadísticas y muestra resultados en términos de eficiencia energética.

4.2.1. Estructura de enerGyPU

enerGyPU es un monitor tipo *batch* dividido en dos niveles [8]. Un nivel de captura de datos en tiempo de ejecución, conformado por tres actividades; y un nivel separado que puede ser utilizado más tarde para visibilizar datos online, conformado por cuatro actividades, como se presenta en la figura 4.3.

Figura 4.3: Estructura del Monitor enerGyPU.



El primer nivel de enerGyPU llamado *Data Capture in Runtime*, es utilizado para establecer el orden de ejecución y declarar las variables de entorno necesarias en el *script* llamado *enerGyPUrun.sh*, para ejecutar las aplicaciones e iniciar la captura de parametros y factores arquitecturales. El LINPACK Benchmark como caso de estudio requiere declarar las variables de entorno de CUDA y MPI: *export LD_LIBRARY_PATH=:/usr/local/cuda-6.5* y *export MPI=:/local/intel/impi/* respectivamente, para que las libreras BLAS de Intel® y NVIDIA®, para interceptar las llamadas al DGEMM y al DTRSM y procesar simultaneamente sobre la CPU y GPU. Además, el usuario escribe los mismos parámetros LINPACK definidos en el archivo *HPL.dat* para ejecutar experimento. Estos parámetros son utilizados por el *script enerGyPUrun.sh* para crear una llave primaria llamada *ARGV*, y encadenar todos los eventos siguientes, que identifican de forma única cada experimento o ejecución del LINPACK Benchmark. El evento (1) crea un directorio en un repositorio de datos llamado *testbed*, donde los datos son centralizados por experimento. El evento (2) y (4) son usados para ejecutar una *script* llamado *enerGyPUrecord.sh*, de

manera que, coordina tareas especiales escritas en AWK, C y las variables de entorno, para procesamiento de texto y colección de datos en el repositorio *testbet*. *enerGyPUrecord.sh* realiza la extracción de datos y genera un archivo separado por cada GPU, para registar en cada segundo el estado de factores computacionales como: la frecuencia de reloj del (*Streaming Multiprocessor*, la frecuencia de reloj de la memoria, el uso de memoria y el consumo de potencia. El evento (3) y (5) inicia el *script run_linpack.sh* para procesar el sistema de ecuaciones lineales, el cual calcula el rendimiento obtenido por el experimento y registra los resultados en su directorio activo en *testbed*, una vez haya terminado la ejecución.

El segundo nivel de enerGyPU llamado *Data visualization Online*, es utilizado en una etapa de pos-procesamiento de la aplicación para realizar los análisis de rendimiento y consumo de potencia por experimento y por conjunto de experimentos de una misma aplicación. Esto despliega información a través de gráficos de linea de tiempo, tablas estadísticas, histogramas y muestra resultados en términos de eficiencia energética. El evento (6) despliega información del experimento a través de IPython Notebook Viewer en un programa llamado *enerGyPUdisplay.ipynb*, el cual contiene rutinas de códigos predefinidos escritos en Python, donde lo usuarios deben de ingresar el argumento *ARGV* para identificar cada experimento por aplicación. *enerGyPUdisplay.ipynb* mostrará los gráficos de linea de tiempo para los factores computacionales por cada GPU en el nodo o maquina. En este punto, el monitor realiza una validación de datos para asegurar la integridad de estos y continuar con el despliegue de tablas estadísticas, gráficos de barras con todos los datos registrados por cada GPU en cada nodo. Posteriormente enerGyPU muestra los resultados de la ejecución de aplicaciones, en términos de: el tiempo de ejecución, el rendimiento computacional, el consumo de potencia, el consumo de energía y el rendimiento por *Watt* Finalmente enerGyPU presenta una comparación entre el consumo de energía utilizado para ejecutar la aplicación y el consumo de energía del nodo cuando está inactivo, en el mismo intervalo de tiempo.

4.2.2. Procedimiento Experimental: Utilización de enerGyPU

Esta sección presenta el uso de enerGyPU para capturar y visualizar los factores computacionales para una ejecución del LINPACK Benchmark. Este experimento fue procesado en un nodo de la configuración 'A' del *cluster* GUANE, donde se registran los factores computacionales para las 8 GPU del nodo.

En este procedimiento experimental utilizamos la versión HPL.2.0 configurada por Tesla GPUs [26]. los parámetros del LINPACK Benchmark que se utilizaron se presentan en el Cuadro 4.1. Estos fueron escritos en el archivo $HPL.dat$ y el script *enerGyPUrun.sh* para ejecutar el experimento. En consecuencia, enerGyPU inicia el procesamiento de texto y centralización de la colección de datos en el repositorio *testbed*, y vincula todos los eventos a una misma llave $ARGV$, la cual representa una estructura única con los parámetros utilizados para este test. El *script enerGyPUrecord.sh* registra los resultados de ejecución

en el repositorio *testbet* cuando finaliza.

Cuadro 4.1: Parametros Globales del Linpack benchmark en uno de los experimentos.

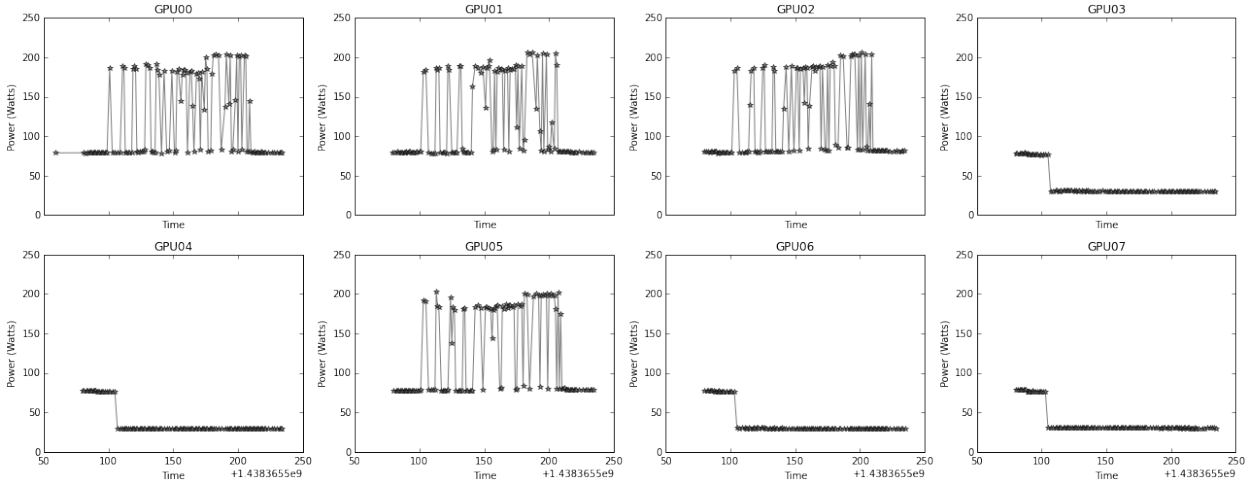
Matrix size	49152
Block size	1024
GPU Used	4
Cores per GPU	3
Process MPI	4

Después de capturar los datos, se crea una sesión interactiva via *tunneling ssh* con el *cluster* GUANE para ejecutar un IPython Notebook Viewer remoto del segundo nivel de enerGyPU o mediante una sesionde Jupyter-hub. Una vez que el programa *enerGyPU-display.ipynb* es iniciado, el usuario debe ingresar el argumento *ARGV*, para identificar el directorio con los datos de ejecución para este procedimiento experimental. Abajo se muestra ocho gráficos de secuencia con los datos (SM Clock Frequency, Memory Clock Frequency, Memory Usage, Power Draw) para cada GPU de un nodo. En la figura 4.4, nosotros mostramos la secuencia de gráficos para el Consumo de Potencia de cada GPU, habilitándonos para observar cual GPUs trabaja mientras se ejecuta Linpack. Después, enerGyPU muestra las estadísticas para analizar los datos que determinan la eficiencia energética como se muestra en la Table 4.2. Este procedimiento experimental muestra 4 GPUs que son inactivadas al promedio de frecuencia de 212MHz para SM clock y 340MHz para la frecuencia de memoria, con el propósito de disminuir el consumo de poder. Las otras 4 GPUs esta trabajando con una frecuencia constante de 1147MHz for SM clock y 1566MHz para la frecuencia de memoria, con una constante de memoria utilizada de 2128MiB, generando un promedio de potencia de 120 watts entre las GPUs trabajando. La desviación estándar de 51 watts muestra que la ejecución puede ser mejorada, por ejemplo, si nosotros incrementamos el número de núcleos por GPU o el número de procesadores MPI para hacer una utilización completa de núcleos GPUs.

Subsecuentemente enerGyPU muestra la cantidad de energía utilizada por cada GPU entre *idlepower* y *runpower*, en términos de Joules. Potencia inactiva es el consumo de potencia cuando una GPU esta encendido pero no tiene aplicaciones corriendo, para esto nosotros encontramos una potencia promedio de consumo de 78.82 Watts para todas las GPUs en el nodo. Con las 4 GPUs que son inactivadas el SM y frecuencia de memoria, como se muestra en Figure 4.5.

El Cuadro 4.3 muestra el resultado obtenido para la ejecución, donde enerGyPU presento 691.20 GFLOPS y muestra los resultados obtenidos para ejecución, donde enerGyPU identifica 691.20 GFLOPS y gasta un tiempo resolviendo el Linpack, y calcula una latencia entre el tiempo de inicio para capturar los datos y el tiempo en que enerGyPU inicia Linpack. Cuando se esta utilizando 4 GPUs, con 3 núcleos por GPU y 4 procesadores

Figura 4.4: enerGyPU - Gráfico del consumo de potencia a través del tiempo.



Cuadro 4.2: enerGyPU - Modulo de estadisticas para analizar los factores que determinan la eficiencia energetica.

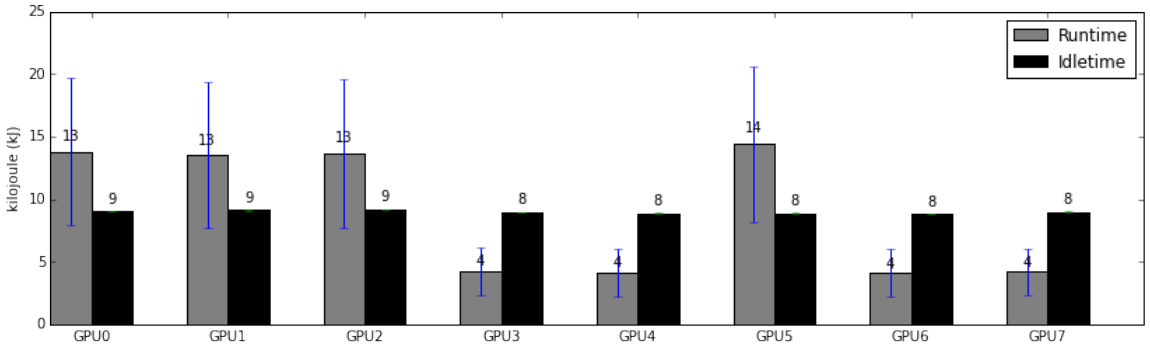
Factors	GPU00	GPU01	GPU02	GPU03	GPU04	GPU05	GPU06	GPU007
SM(MHz) Mean	1147	1147	1147	215.74	223.73	1147	204.66	206.65
SM(MHz) Std	0	0	0	282.46	301.54	0	262.64	268.84
Memory(MHz) Mean	1566	1566	1566	346.53	346.53	1566	334.09	334.09
Memory(MHz) Std	0	0	0	507.90	507.90	0	495.24	495.24
MEM Usage(MiB) Mean	2128.9	2128.89	2128.9	10	10	2128.87	10	10
MEM Usage(MiB) Std	0.30	0.31	0.30	0	0	0.33	0	0
Power(Watts) Mean	120.55	118.01	119.12	37.22	36.27	125.65	36.07	36.76
Power(Watts) Std	51.29	50.92	51.36	16.62	16.78	54.32	16.26	16.00
Energy(KJ) Mean	13.80	13.51	13.64	4.26	4.15	14.39	4.13	4.21
Energy(KJ) Std	5.87	5.83	5.88	1.90	1.92	6.22	1.86	1.83

MPI para resolver un sistema de ecuaciones lineales de 49152∗49152, un nodo de cluster GUANE alcanzó la eficiencia energetica de 1097.68 MFLOPS/W.

4.3. Evaluación de Rendimiento y Consumo de Potencia

Esta evaluación de rendimiento y consumo de potencia utiliza una colección de resultados obtenidos a través de enerGyPU al ejecutar una combinación de experimentos del LINPACK Benchmark. Los parametros fueron seleccionados siguiendo un diseño fraccionado factorial propuesto por Raj Jain [8] serán descrito en la próxima subsección.

Figura 4.5: enerGyPU - Gráfico de barras para analizar el consumo de energía entre las GPU en estado Idletime and Runtime.



enerGyPU Time:	120 sec
enerGyPU Latency:	6 sec
HPL2.0 Time:	113,96 sec
HPL2.0 Performance:	694,7 GFLOPS
Power Consumption:	539,89 Watts
Performance per Watt:	1286,76 MFLOPS/W
Energy Efficient:	**61.52 kJ**

Cuadro 4.3: eGPU-Results to analyzes of Linpack benchmark in this experimental procedure.

4.3.1. Diseño Experimental para HPL-2.0 sobre GUANE

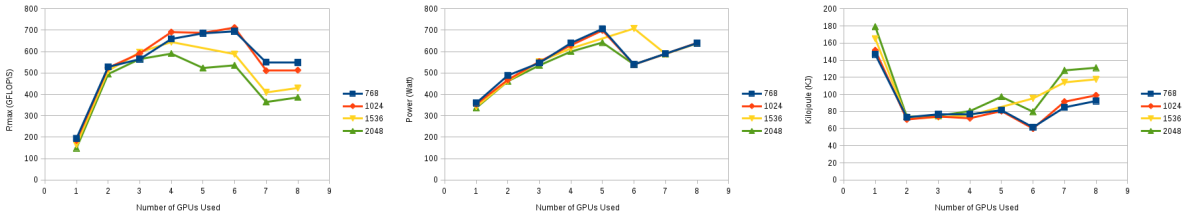
El experimento consiste en: 12 diferentes parámetros de carga de trabajo usando tres matrices de tamaño *(49152,61440,73728)* con cuatro bloques de tamaño *(768,1024,1536,2048)*, para dividir estas matrices. 8 diferentes combinaciones de parámetros de arquitectura como de GPU, número de *cores* para procesamiento, número de *cores* para administración de GPU y número de procesadores MPI. 2 factores de control (frecuencia de reloj del SM y de la memoria) que representan los tres niveles de poder para el GPU *worker*, GPU *idle* y GPU *baseline* o estado base.

4.3.2. Resultados de HPL-2.0 sobre GUANE

La combinación arquitectural de {6 GPU, 2 cores por GPU and 6 MPI process es el recurso computacional más energéticamente eficiente utilizado en este experimento para resolver una matriz de tamaño 49142, dividida en bloques de tamaño 768, los cuales generaron 48 tareas con $384 Mbytes$ de granularidad de tareas como se presenta en la

Figura 4.6. Este experimento ejecutó el HPL-2.0 en 113,96sec con un promedio de consumo de potencia por nodo de 539,87$Watts$ para obtener un rendimiento computacional 694,7$GFLOPS$ y requierió un total de 61,52$KJoules$ como energía para solución.

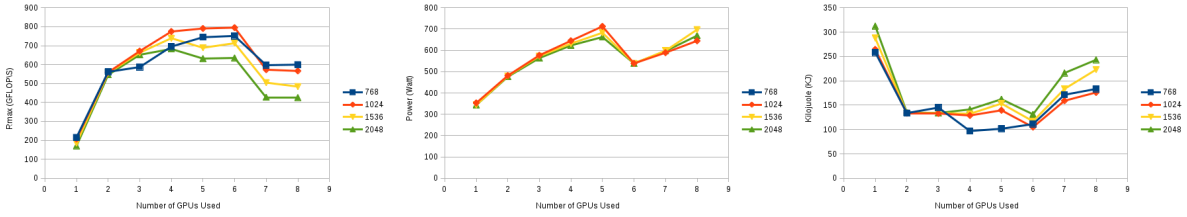
Figura 4.6: Resultados de Resolver el DGEMM con una Matrix de 49142 en GUANE



(a) Rendimiento (b) Consumo de Potencia (c) Consumo de Energía

La combinación arquitectural de {4 GPU, 3 cores por GPU and 4 MPI process} es el recurso computacional más energéticamente eficiente utilizado en este experimento para resolver una matriz de tamaño 61440, divida en bloques de tamaño 768, los cuaes generan 80 tareas con 360$Mbytes$ de granularidad de tareas como se muestra en la Figura 4.7. Este experimento ejecutó el HPL-2.0 en 222,47sec con un promedio de consumo de potencia de 435,06$Watts$ para obtener un rendimiento computacional de 695$GFLOPS$ y requierió un total de 96,7$KJoules$ como energía para solución.

Figura 4.7: Resultados de Resolver el DGEMM con una Matrix de 61440 en GUANE

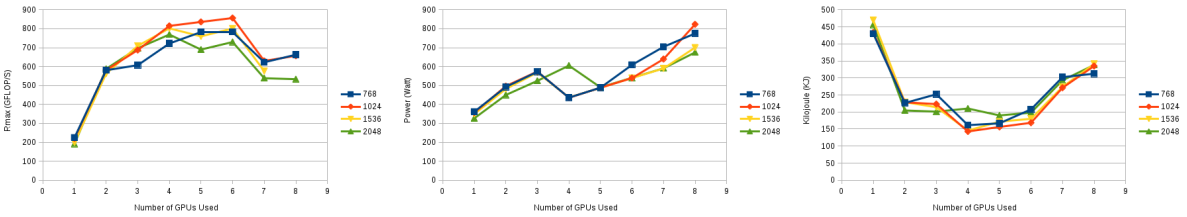


(a) Rendimiento (b) Consumo de Potencia (c) Consumo de Energía

La combinación arquitectural de {4 GPU, 3 cores por GPU and 4 MPI process} es el recurso computacional más energéticamente eficiente utilizado en este experimento para resolver una matriz de tamaño 73728, divida en bloques de tamaño 1024, los cuales generan 72 tareas con 576$Mbytes$ de granularidad de tareas como se muestra en la Figura 4.8. Este experimento ejecutó el HPL-2.0 en 327,9sec con un promedio de consumo de pontencia por nodo de 435,43$Watts$ para obtener un rendimiento computacional de 814,8$GFLOPS$ y requierió un total de 142,77$KJoules$ como energía para solución.

Los resultados de enerGyPU más representativos para resolver diferentes matrices, variando el tamaño de entrada al HPL-2.0 en GUANE son presentados en la tabla 4.4.

Figura 4.8: Resultados de Resolver el DGEMM con una Matrix de 73728 en GUANE



(a) Rendimiento (b) Consumo de Potencia (c) Consumo de Energía

Esta tabla esta clasificada en: la mejor energía para solución (ES), el peor consumo de potencia (PC) y el peor tiempo de ejecución (ET) para cada tamaño de entrada de la matriz. Las dos últimas clasificaciones representan la peor energía para solución divida en dos comportamientos: El primero es dado por el alto consumo de potencia determinado por el uso de todas las GPU y la segunda es dada por una ejecución larga en el tiempo, determinada por el uso de mínimos recursos computacionales.

4.3.3. Niveles de Potencia sobre GUANE

La utilización de enerGyPU variando las cargas de trabajo y parámetros de arquitectura, para caracterizar las ejecuciones del HPL-2.0 al procesar diferentes tamaños de matrices sobre los nodos del *clusters* GUANE. Permitió reconocer los niveles de potencia de las GPU para cada combinación de GPU a utilizar en el procesamiento. El consumo de potencia total está representado por dos estados: *GPU worker* y *GPU idle*, cuando se usa técnicas de *power capping*.

GPU worker: Representa el número de GPU que el desarrollador seleccionó para realizar tareas de procesamiento, la cual usa el nivel máximo de power capping con una frecuencia constante de 1145 *MHz* para SM clock y 1566 *Mhz* para memoria reloj, con una constante de uso de memoria de 2128 *MiB*, esto genera un promedio de potencia de 1120 *Watts* para cada GPU trabajando.

GPU idle: Representa el número de GPU que se encuentran inactivas, las cuales utilizan el minimo nivel de potencia con una constante de frecuencia de 212 *MhZ* para SM clock y 340 *MHz* para memoria reloj, con un 0 % de memoria utilizada para reducir el consumo de poder a 29.46 *Watts*.

Cuadro 4.4: Resultados representativos de enerGyPU para analizar el mejor experimento para *"energetic to solution:* **energía de solución"** (ES), el experimento con el mayor *"power consumption:* **consumo de potencia"** (PC) y el mayor *"execution time:* **tiempo de ejecución"** (ET), para resolver cada matriz.

Experiments to solve HPL2.0	*ES*	*PC*	*ET*	*ES*	*PC*	*ET*	*ES*	*PC*	*ET*
Global Workload Parameters									
Ms	49152			61440			73728		
Bs	768	2048	2048	768	2048	2048	1024	1536	1536
Dependent Parameters									
$Task_{Size}$ *(MBytes)*	384	768	768	360	960	960	576	864	864
$Task_{Number}$	48	24	24	80	30	30	72	48	48
Architectural Parameters for Nvidia GPU									
GPU_{Usage}	6	8	1	4	8	1	4	8	1
$Cores_{GPU}$	2	1	12	3	1	12	3	1	12
MPI_{GPU}	6	8	1	4	8	1	4	8	1
enerGyPU Results									
$enerGyPU_{Time}$ *(sec)*	120	211	537	228	369	915	333	493	1350
$enerGyPU_{L}atency$ *(sec)*	6	6	5	6	6	6	6	6	6
$HPL2{,}0_{Time}$ *(sec)*	113.96	205.23	531.31	222.47	363.28	909.34	327.9	486.76	1344.09
Rmax (GFLOPS)	694.7	131.10	149	695	425.6	170	814.8	548.9	198.8
$Power_{GPU}$ *(Watts)*	539.87	638	337.16	435.06	669.28	337.16	435.43	700.63	337.16
PPW (MFLOPS/W)	1286.79	603.93	441.92	1597.44	635.90	494.85	1871.22	783.43	569.37
$E_{Solution}$ ***(KJ)***	**61.52**	**131.10**	**179.13**	**96.7**	**243.13**	**312.38**	**142.77**	**341.04**	**469.25**

4.4. Conclusion

Construimos enerGyPU para facilitar la captura, visualización de datos y para analizar multiples ejecuciones bajo diferentes combinaciones de parámetros, por tanto, observar la granuralidad de los factores que determinan la eficiencia energética en los nodos del *cluster* GUANE. Una de las caracteristicas de constucción de enerGyPU es permitir la monitorización de aplicaciones compiladas previamente donde los investigadores no necesitan crear identificadores en el código para ejecutar enerGyPU, asegurando la integridad de los resultados. Basados en los procedimientos de experimentación y resultados presentados, enerGyPU es una buena alternativa para analizar el consumo de potencia en *clusters* con multi-GPU a un nivel de software y puede ser complementado con otros monitores de energía que estan diseñados para ser conectados directamente a la potencia permitiendo medidas holisticas en clusters con multi-GPUs.

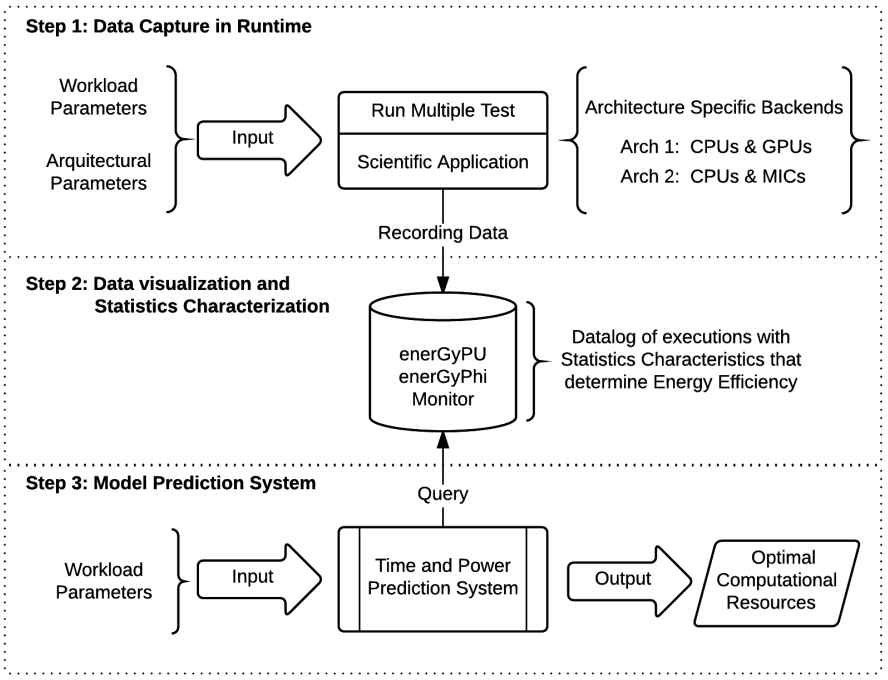
Capítulo 5

Predicción del Tiempo y Potencia para un Aceleramiento Energéticamente Eficiente (AEE)

5.1. Esquema Integrado de Aceleración Energéticamente Eficiente (AEE)

Esta investigación propone un esquema integrado *energy-aware* llamado Aceleración Energéticamente Eficiente (AEE) para aplicaciones científicas de gran escala sobre arquitecturas heterogéneas. El esquema AEE tiene un flujo de trabajo en tres niveles como se muestra en la Figura 5.1. El primer nivel llamado *"Data Capture in Runtime:* **Captura de Datos en Tiempo de Ejecución"**, permite ejecutar el monitor enerGyPU en paralelo con la aplicación científica, usando diferentes combinaciones de recursos computacionales para capturar factores que determinan la eficiencia energetica como el *streaming multiprocessor*, la frecuencia de reloj, la frecuencia de memoría y el uso de memoria en nodos multi-GPU. El segundo nivel llamado *"Data visualization and Statistics Characterization:* **Visualización y Caracterización Estadistíca"**, utiliza el monitor enerGyPU para analizar los factores claves de los datos obtenidos en cada experimento realizado en el primer nivel, así mismo presenta los resultados obtenidos en términos de eficiencia energética y permite estimar los niveles de potencia para cada supercomputador. El tercer nivel llamado *"Model Prediction System:* **Sistema de Predicción del Modelo AEE"**, utiliza la base de datos *"testbed"* del monitor enerGyPU para ejecutar y modelar el sistema de predicción del tiempo y potencia, para predecir los recursos computacionales óptimos que minimizan el tiempo y el consumo de energía a la vez que maximizan el rendimiento computacional. Esta proyección se presenta en un tiempo estático para mapear la granularidad de tareas de la aplicación científica sobre arquitecturas heterogéneas.

Figura 5.1: Esquema de Aceleración Energéticamente Eficiente (AEE)



5.1.1. Nivel 1: Captura de Datos en Tiempo de Ejecución

En el primer nivel se escogen los parametros globales de la carga de trabajo y de la arquitectura computacional de acuerdo a la combinación de recursos que se esté utilizando (CPU, GPU y MIC). Los investigadores deben diseñar experimentos completos o factoriales de acuerdo a la metodología seleccionada en [8]. Estos experimentos se pueden realizar con diferentes tamaños de datos como entrada (de acuerdo al tamaño de la memoria principal del sistema) y deben tener diferentes combinaciones de recursos computacionales y para capturar factores, como la frecuencia del *Streaming Multiprocessor*, la frecuencia de memoria de la GPU, el uso de memoria, el número de GPU, usando técnicas de *power capping* para ahorrar consumo de energía en los tiempos de transferencia de datos. Los datos de entrada y numero de GPUs son los parametros principales para determinar el numero total de tareas y el tamaño en *MBytes*. De acuerdo a los resultados de está investigación la mejor distribución de tareas, está condicionada al tamaño del bus *PCIe* y a la cantidad de GPU que comparten el bus, por lo tanto las distribuciones de combinaciones de GPUs que presentan mejores resultados de eficiéncia energetica, son:

- Usar *dos GPU* para procesar tareas con grandes tamaños de datos (768*MBytes* - 576*MBytes*) y (números pequeños de tareas (24 - 32).
- Usar *cuatro GPU* para procesar tareas con tamaños medios de datos (576*MBytes* - 432*MBytes*) y números grandes de tareas (76 - 96).

- Usar *seis GPU* para procesar tareas con tamaños medios de datos (360*MBytes* - 288*MBytes*) y números grandes de tareas (64 e 80).

5.1.2. Nivel 2: Visualización y Caracterizacion Estadistíca

El segundo nivel utiliza el monitor enerGyPU en el pos-procesamiento para la visualización de datos y la caracterización estadistíca de cada arquitectura. Este monitor despliega información vía secuencia de datos, tablas estadisticas, histogramas y muestra resultados en terminos de eficiencia energetica, para cada experimento. La caracterización

Una de las técnicas ampliamene utilizadas para la **"calendarización de tareas** *task scheduling"* son los **grafos acíclicos dirigido** *directed acyclic graph (DAG)"*. Así mismo, cada método para resolver sistemas de ecuaciones lineales y diferenciales, presentan una gran variedad de granularidad de tareas, donde varía la tasa de comunicación de datos para compartir las condiciones de frontera de cada problema, como lo describe el trabajo [27]. Este trabajo usa el HPL, el cual es una implementación de la decomposición LU para resolver un sistemas lineales densos $N * N$ de ecuaciones como caso de estudio.

5.1.3. Nivel 3: Modelo de Predicción del Tiempo y Potencia

El tercer nivel utiliza la base de datos de todas las ejecuciones capturadas a través del monitor enerGyPU, para modelar la distribución de tareas, las regresiones lineales y multivariables con los parametros globales y de arquitectura, como se muestra en las Figuras 5.4, 5.5, 5.6. Se desarrolló un sistema de predicción para la Aceleración Energéticamente Eficiente AEE en Python, el cual utiliza los resultados proyectados del modelo de regresión multivariable, para conocer metricas como: el tiempo, el rendimiento, el consumo de potencia, el consumo de energía y el rendimiento por *watt*, que son utilizadas para ejecutar el HPL por cada combinación de recursos computacionales y así calcular la mejor combinación de recursos computacionales, como se muestra en la Figura 5.2.

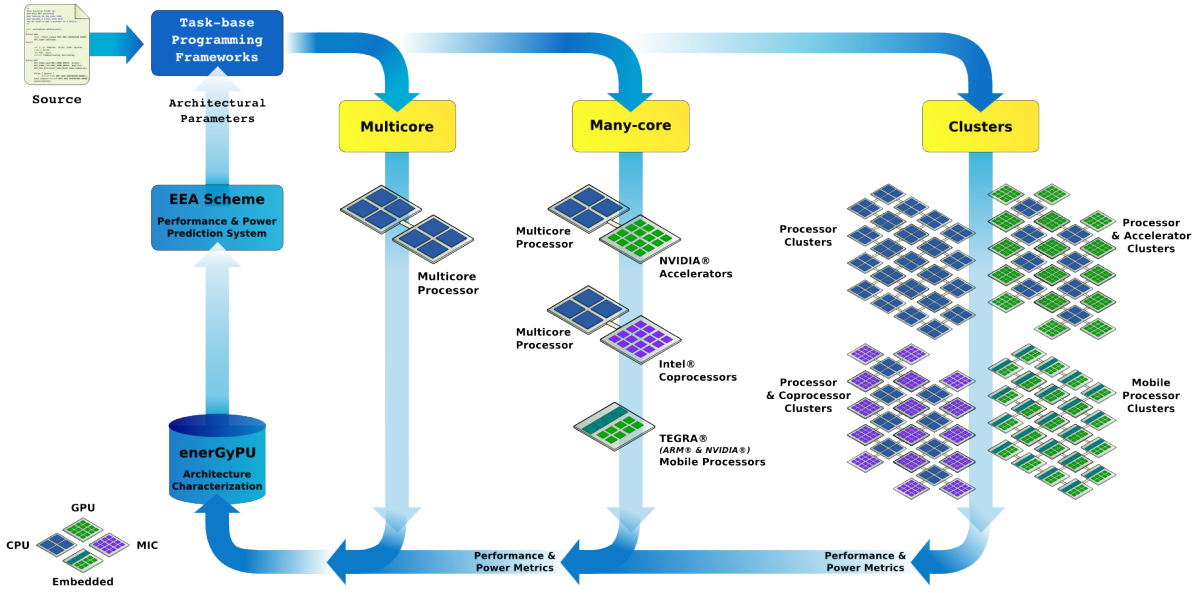
Figura 5.2: Ejemplo de uso del sistema de predicción AEE de una matriz con tamaño 49152 para ejecutar el HPL-2.0 sobre GUANE

	Best Prediction	Time_to_Solution	Performance_Rmax	Power_Consumption	Energy_to_Solution	Performance_per_Watt
block_size	768.000000	1024.000000	768.000000	2048.000000	1024.000000	768.000000
Number_task	64.000000	48.000000	64.000000	24.000000	48.000000	64.000000
Task_size	288.000000	384.000000	288.000000	768.000000	384.000000	288.000000
GPU	3.000000	5.000000	5.000000	1.000000	3.000000	3.000000
Cores	4.000000	2.000000	2.000000	12.000000	4.000000	4.000000
Time_to_Solution	94.217538	12.824716	58.914776	729.964183	48.127479	94.217538
Performance_Rmax	661.631583	639.578050	673.696532	120.236949	627.513101	661.631583
Power_Consumption	546.295967	610.388860	607.907019	356.836346	548.777808	546.295967
Energy_to_Solution	55.262744	40.403046	64.226083	241.981202	31.439707	55.262744
Performance_per_Watt	1.220739	1.079443	1.153953	0.399236	1.146229	1.220739

5.2. Sistema de Predicción de Tiempo y Potencia (AEE)

Como se muestra en la Figura 5.3, la evaluación de rendimiento y el consumo de potencia es un paso clave en el diseño de aplicaciones para grandes sistemas computacionales, como supercomputadores y *clusters* (con nodos *multicore* y aceleradores gráficos, nodos *multicore* y coprocesadores, nodos *manycore* y aceleradores gráficos). Los desarrolladores deben diseñar varios experimentos para caracterizar la carga de trabajo en cada supercomputador, observando las implicaciones arquitectónicas al utilizar diferentes combinaciones de recursos computacionales: como el número de *cores* para procesamiento, el numero de *cores* para administración de GPUs, el número de GPUs para procesamiento, el numero de procesos MPI y las politicas de afinidad de los *threads*. También se debe acoplar factores como la frecuencia de reloj, el uso de memoria y el ancho de banda, para seleccionar la combinación de recursos computacionales que aumenta el rendimiento y minimiza el consumo de potencia al procesar una aplicación de gran escala.

Figura 5.3: Organización de procesadores y coprocesadores en diferentes maquinas computacionales



Adaptado de **James R.** Intel Xeon processors and Intel Xeon Phi coprocessors, 2012

5.2.1. Factores y Parametros Globales

Los parámetros del esquema EEA son utilizados por el monitores enerGyPU, para la captura de datos en tiempo de ejecución, mientras se ejecuta en paralelo con el código HPL variantes en los nodos del clúster Guane y Phi-servidor.

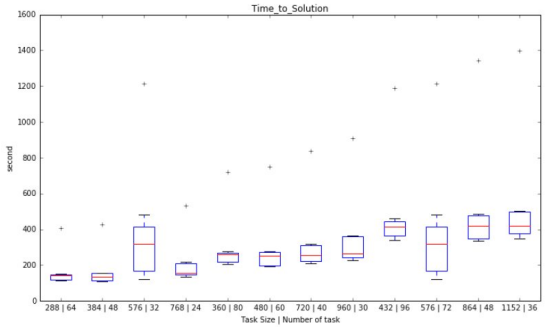
Los parámetros globales se utilizan para ajustar la carga de trabajo de un tamaño problema de la matriz dividida en bloques para generar la tarea total para el procesamiento de la $Task_{Number}$ y obtener el $Task_{size}$ de tamaño en cada experimento. Los parámetros arquitectónicos son la combinación de los recursos de cálculo y configuraciones utilizadas para el procesamiento de las variantes del código de HPL. Los factores de control están monitoreando para analizar las implicaciones en el consumo de energía y el rendimiento cuando se utilizan diferentes combinaciones de recursos computacionales para resolver el problema de HPL. Los parámetros dependientes se podrían utilizar para crear un modelo que prediga el consumo de tiempo y de energía que determinan la eficiencia energética en un tamaño específico problema, que se describe en la Cuadro 5.1.

Cuadro 5.1: EEA parameters used by enerGyPU and enerGyPhi

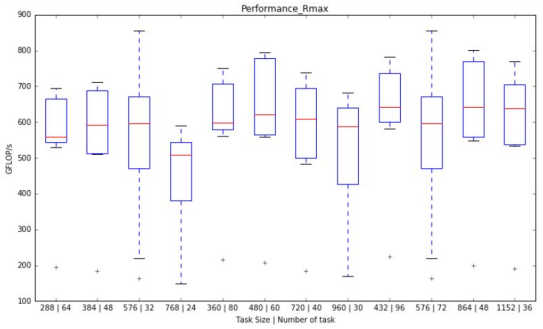
Global Workload Parameters	
Ms	Matriz size
Bs	Block size
Architectural Parameters for Nvidia GPU	
GPU_{Usage}	Number of GPU used to HPL
$Cores_{GPU}$	Number of cores per GPU to HPL
MPI_{GPU}	Number MPI process same of GPU used
Architectural Parameters for Intel Xeon Phi	
$Cores_{MIC}$	Number of cores used on MIC
$Threads_{MIC}$	Number of threads per core on MIC
Control Factors for Nvidia Tesla GPU	
SM_{Clock}	GPU SM clock frequency
Mem_{Clock}	GPU memory clock frequency
Control Factors for Intel Xeon Phi	
$Thread_{Affinity}$	Threads execution policy on MIC cores
$Thread_{Granularity}$	Hyperthreading used per core
Dependent Parameters	
$Task_{Size}$	Bytes allocated per task
$Task_{Number}$	Total task for processing
$GPUMem_{Usage}$	GPU memory usage
$MICMem_{Usage}$	MIC memory usage
$Time$	Total time execution of application
$Power_{GPU}$	Total power consumption by all GPU
$Power_{MIC}$	Total power consumption by MIC

Figura 5.4: Distribución de Tareas a Partir de los Resultados de enerGyPU

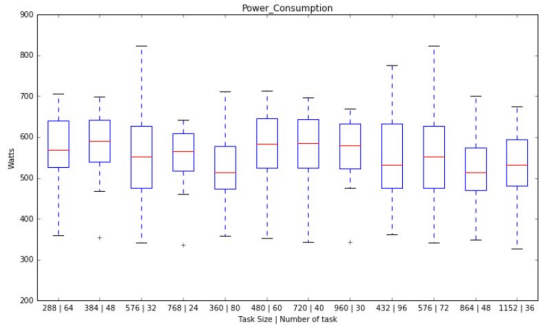
(a) Variabilidad de las Tareas Tiempo al Ejecutar el HPL-2.0 Sobre GUANE



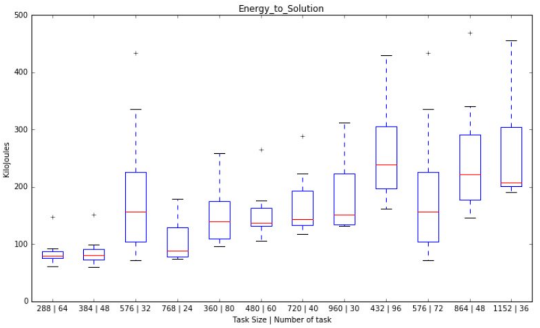
(b) Variabilidad de las Tareas Rendimiento al Ejecutar el HPL-2.0 Sobre GUANE



(c) Variabilidad de las Tareas Consumo de Potencia al Ejecutar el HPL-2.0 Sobre GUANE



(d) Variabilidad de las Tareas Energía al Ejecutar el HPL-2.0 Sobre GUANE



1

(e) Variabilidad de las Tareas PPW al Ejecutar el HPL-2.0 Sobre GUANE

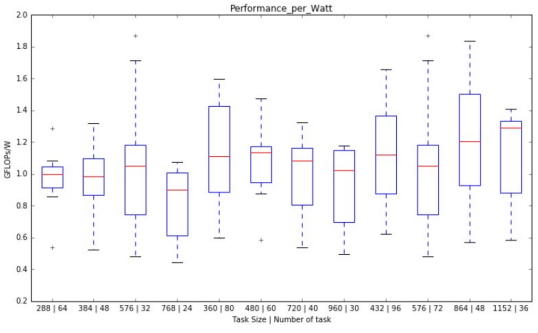
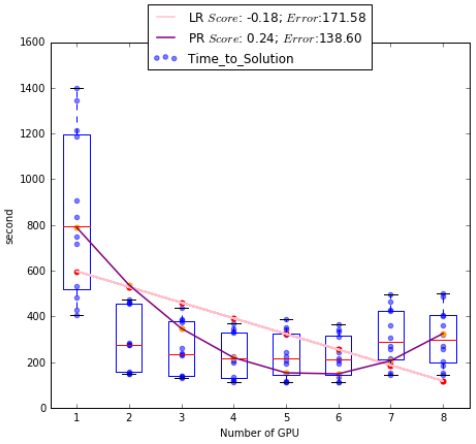
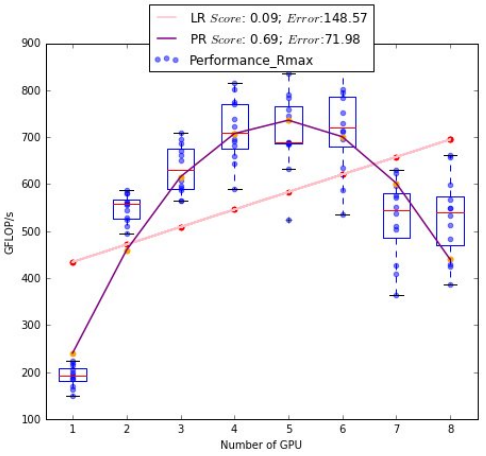


Figura 5.5: Modelo Lineal Utilizando el Número de GPU

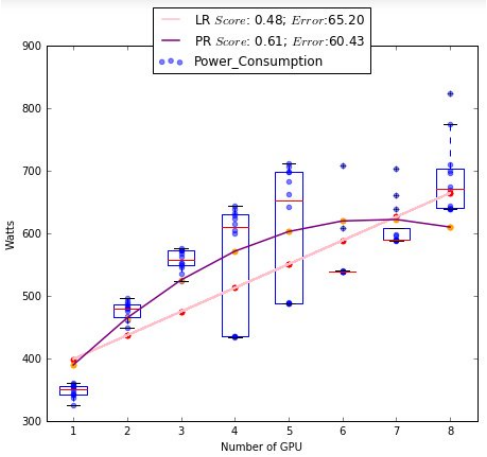
(a) Regresión Lineal del Tiempo para Ejecutar el HPL-2.0 Sobre GUANE



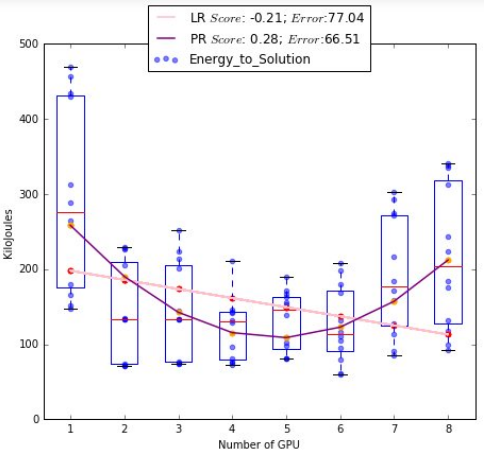
(b) Regresión Lineal del Rendimiento para Ejecutar el HPL-2.0 Sobre GUANE



(c) Regresión Lineal del Consumo de Potencia para Ejecutar el HPL-2.0 Sobre GUANE



(d) Regresión Lineal del Consumo de Energía para Ejecutar el HPL-2.0 Sobre GUANE



(e) Regresión Lineal del PPW para Ejecutar el HPL-2.0 Sobre GUANE

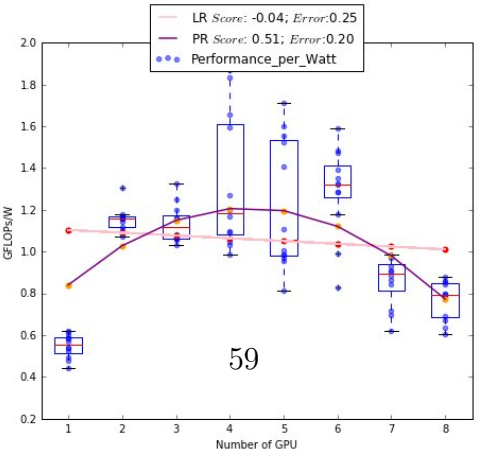
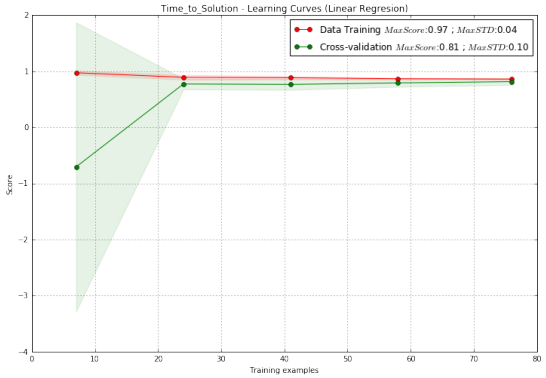
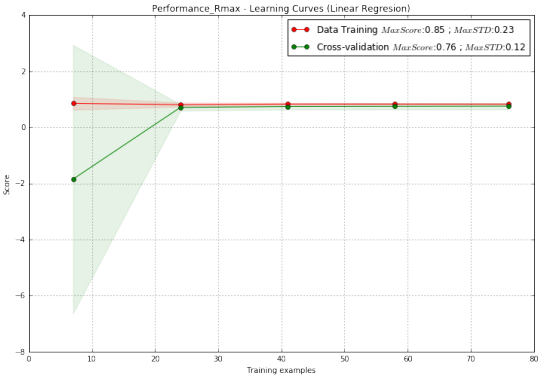


Figura 5.6: Modelo Multivariable con los Parametros Globales y de Arquitectura

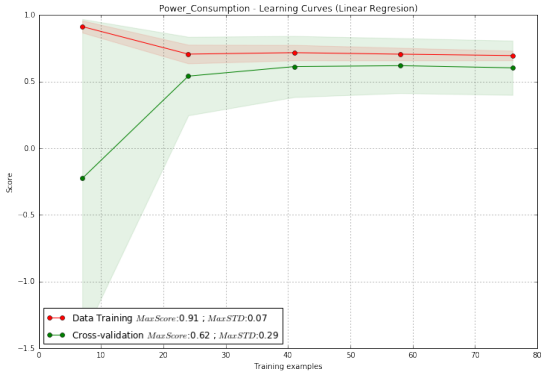
(a) Regresión Multivariable del Tiempo para Ejecutar el HPL-2.0 Sobre GUANE



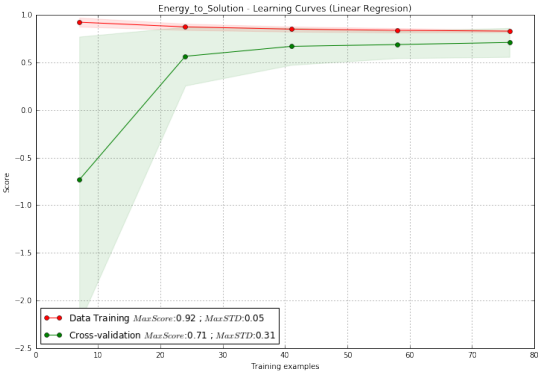
(b) Regresión Multivariable del Rendimiento para Ejecutar el HPL-2.0 Sobre GUANE



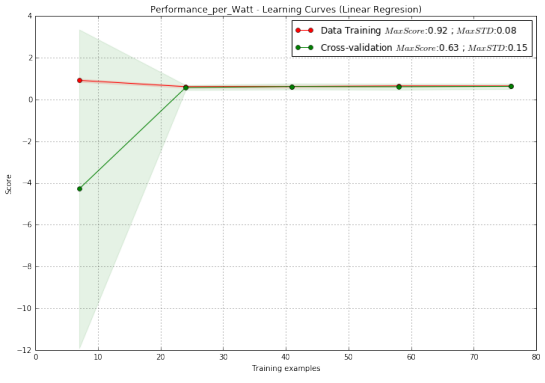
(c) Regresión Multivariable del Consumo de Potencia para Ejecutar el HPL-2.0 Sobre GUANE



(d) Regresión Multivariable del Consumo de Energía para Ejecutar el HPL-2.0 Sobre GUANE



(e) Regresión Multivariable del PPW para Ejecutar el HPL-2.0 Sobre GUANE



Capítulo 6

Conclusiones

6.1. Discusión

6.1.1. Power Capping y Balanceo de Cargas para CPU y GPU

Power Capping es una técnica ampliamente utilizada para minimizar el consumo de energía en los supercomputadores, a través de la regulación de frecuencia y voltajes dinámico (DVFS), esta técnica permite ajustar manual o automáticamente la frecuencia y el voltaje de la CPU y recientemente en la GPU. Komoda et al. [28] utiliza tecnicas de *power capping* para administrar las frecuencias de aceleradores gráficos y mapeo de tareas al comienzo de la ejecución, así mismo, presenta un modelo empírico con un pequeño número de perfiles para predecir el tiempo de ejecución, el consumo de energía y el uso de la frecuencia de la CPU y la GPU, como parámetros principales. El framework GreenGPU propone una regulación de potencia y balanceo de cargas en dos niveles, el primer nivel: divide dinámicamente y distribuye las cargas de trabajo en la CPU y en la GPU tratando que finalicen al mismo tiempo. El segundo nivel de GreenGPU: regula dinámicamente las frecuencias de los núcleos de la GPU y la memoria de una manera coordinada, en función de sus utilizaciones [14]. Estos esquemas de limitación de energía y equilibrio de carga funcionan bien cuando se utiliza una CPU y una GPU; sin embargo, al tener muchos nodos con multi-GPU (4, 6 y 8 GPU) el problema se transfiere a saber la cantidad de recursos computacionales que se necesitan para mapear las tareas de una carga de trabajo específica. Hogn and Wang [2] propone un modelo de predicción integrada de potencia (IPP) para arquitecturas GPU. IPP predice el rendimiento por *watt* y también el número óptimo de núcleos para ahorrar energía en una GPU. El número de núcleos activos predichos por IPP sólo involucra el número de bloques dentro de una aplicación, a menos que cambien el hardware o el programador de subprocesos.

6.1.2. Esquema Energy-aware y Modelamiento de Potencia para Clusters

Huo et al. [29] presenta un esquema de programación de tareas eficiente de energía para balancear las cargas de trabajo y regular el consumo de potencia en los *clusters* heterogéneos tipo CPU-GPU. De acuerdo con la política seleccionada en el nodo, basado en el historial de utilización de la GPU, se puede disminuir el consumo de energía estática de la GPU en estado de reposo. Este método de evaluación del uso de esquema de número de la base GPU adecuado para una tarea particular en función del rendimiento por vatio ha sido propuesto por Zhang et al. [30] heurística greedly que puede implicar los pesos del procesador a la cartografía de aproximadamente óptima entre las tareas y los recursos computacionales. Particionan todas las tareas en seis tipos de clasificación basadas en el grado de paralelismo, la carga de trabajo y el tamaño de los datos de entrada de la tarea. Los datos de entrada es un parámetro clave para determinar el número total de *threads* y la carga de trabajo es un parámetro clave para la calendarización de tareas. El algoritmo WSLSA presenta problemas de eficiencia cuando las tareas grandes de la cargas de trabajo son demasiadas, de acuerdo a sus politicas de asignación de tareas; sin embargo, es posible sintonizar el algoritmo WSLSA. Sîrbu y Babaoğlu [31] presentan un estudio sobre la modelización y predicción del consumo de energía en un superordenador híbrido de CPU-GPU-MIC. Utilizando datos de un marco de seguimiento dedicado, que construyen un modelo basado en datos de consumo de energía para cada usuario en el sistema y lo utilizan para predecir los requerimientos de energía de los trabajos futuros. Utilizaron la clase *CountVectorizer* del paquete python *scikit-learn* y el uso de las frecuencias como principales parámetros de potencia para predecir el uso de energía por puesto de trabajo en 5 análisis de regresión, una para cada tipo de componente, y luego sumar los componentes de potencia predios para obtener una estimación de la potencia global de trabajo.

Por lo tanto, esta investigación propone un esquema integrado Aceleración Energéticamente Eficiente (AEE), con un flujo de trabajo en tres niveles: *"Data Capture in Runtime:* **Captura de Datos en Tiempo de Ejecución"**, *"Data visualization and Statistics Characterization:* **Visualización y Caracterización Estadistíca"**, *"Model Prediction System:* **Sistema de Predicción del Modelo AEE"**, para seleccionar los recursos computacionales necesarios, que puede ser utilizados por *frameworks* de balanceo de carga (Ej. StarPU) y programadores Y así, realizar un buen mapeo de tareas en problemas de ecuaciones lineales de gran escala, con regulación de la frecuencia de procesamiento, acelerar el rendimiento computacional de las aplicaciones científicas y reducir el consumo de energía en arquitecturas heterogéneas tipo CPU-GPU.

6.2. Conclusiones

- Este trabajo de investigación determinó: que utilizar un mayor número de tareas con tamaños pequeños de datos, permite una aceleración enérgeticamente eficiente, para procesar sistemas de ecuaciones lineales sobre nodos con arquitecturas heterogéneas tipo CPU-GPU.
- El uso de *frameworks* de balanceo de carga es una herramienta útil, para solucionar el problema de portabilidad en aplicaciones científicas de gran escala y explotar los recursos computacionales sobre nuevas arquitecturas.
- Los procesadores Tegra TK1 de bajo consumo de energía, son una buena alternativa para minimizar el tiempo de transferencia de datos en el procesamiento simultaneo entre CPU@cores y CUDA@cores.
- Se diseño e implementó enerGyPU para facilitar la captura, la visualización de datos y analizar muchos experimentos bajo diferentes combinaciones de parámetros y observar la granularidad de los factores de control que determinan la eficiencia energética, sin necesidad de instrumentar las aplicaciones científicas.
- Se diseñó e implementó el esquema AEE, para seleccionar los recursos computacionales necesarios, que puede ser utilizados pro *frameworks* de balanceo de carga (Ej. StarPU) y programadores Y así, realizar un buen mapeo de tareas en problemas de ecuaciones lineales de gran escala, con regulación de la frecuencia de procesamiento, para acelerar el rendimiento computacional de las aplicaciones y reducir el consumo de energía en arquitecturas heterogéneas tipo CPU-GPU.
- Este trabajo comprueba que es necesario caracterizar el problema, la arquitectura computacional y predecir el número de recursos computacionales, para regular la potencía y realizar un balanceo de cargas enérgeticamente eficiente en cada tipo de arquitectura

Bibliografía

[1] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, and Others. The opportunities and challenges of exascale computing. *summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science*, 2010.

[2] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In André Seznec, Uri C. Weiser, and Ronny Ronen, editors, *ISCA*, pages 280–289. ACM, 2010.

[3] Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Commun. ACM*, 56(2):93–102, 2013.

[4] Prometeus GmbH. TOP500 List.Copyright 1993-2015 TOP500.org (c).

[5] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. Williams, and K. Yelick. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead. Technical report, 2008.

[6] Saša Stojanovic, Dragan Bojić, Miroslav Bojović, Mateo Valero, Veljko Milutinović. An overview of selected hybrid and reconfigurable architectures. University of Belgrade, Serbia, Polytechnic University of Catalonia, Spain, University of Belgrade, Serbia, 2012.

[7] Hernández Sampieri. Roberto. *Metodología de la investigación.* McGraw-Hill., 6ta edition, 2014.

[8] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[9] NVIDIA Corporation. NVIDIA Tesla P100, Whitepaper. The Most Advanced Datacenter Accelerator Ever Built Featuring Pascal GP100 the World's Fastest GPU. 2016. Technical report.

[10] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[11] Cedric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-Andre; Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurr. Comput. : Pract. Exper.*, 2011.

[12] Chun Liu, Anand Sivasubramaniam, Mahmut Kandemir, and Mary Jane Irwin. Exploiting Barriers to Optimize Power Consumption of CMPs. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, IPDPS '05, pages 5.1–, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Jungseob Lee, Vijay Sathisha, Michael Schulte, Katherine Compton, and Nam Sung Kim. Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, PACT '11, pages 111–120, Washington, DC, USA, 2011. IEEE Computer Society.

[14] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *ICPP*, pages 48–57. IEEE Computer Society, 2012.

[15] Edson L. Padoin, Daniel A. G. de Oliveira, Pedro Velho, and Philippe O. A. Navaux. Time-to-Solution and Energy-to-Solution: A Comparison Between ARM and Xeon. In *Proceedings of the 2012 Third Workshop on Applications for Multi-Core Architecture*, WAMCA '12, pages 48–53, Washington, DC, USA, 2012. IEEE Computer Society.

[16] Gustavo Rostirolla, Rodrigo da Rosa Righi, Vinicius Facco Rodrigues, Pedro Velho, and Edson Luiz Padoin. GreenHPC: a novel framework to measure energy consumption on HPC applications. In *SustainIT*, pages 1–8. IEEE, 2015.

[17] Jack Dongarra. Algorithmic and Software Challenges For Numerical Libraries at Exascale, 2013.

[18] Victor Martinez, David Michéa, Fabrice Dupros, Olivier Aumage, Samuel Thibault, Hideo Aochi, and Philippe O. A. Navaux. Towards Seismic Wave Modeling on Heterogeneous Many-Core Architectures Using Task-Based Runtime System. In *27th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2015, Florianópolis, Brazil, October 17-21, 2015*, pages 1–8, 2015.

[19] Victor Martinez, John García, Carlos Barrios, Fabrice Dupros, Hideo Aochi and Navaux Philippe. Task-based programming on low-power Nvidia Jetson TK1 manycore architecture: Application to earthquake modeling. CARLA 2015: Latin American HPC Conference. (Speech), 2015.

[20] Jean Virieux. SH-wave propagation in heterogeneous media; velocity-stress finite-difference method. Geophysics. 51, 889–901, 1986.

[21] Luiz Ramos Luís F. W. Góes Alyson D. Pereira. PSkel: A stencil programming framework for CPU-GPU systems Concurrency and Computation Practice and Experience 27(17):4938-4953, 2015.

[22] Fabrice Dupros, Hiep-Thuan Do, and Hideo Aochi. On Scalability Issues of the Elastodynamics Equations on Multicore Platforms. In *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, pages 1226–1234, 2013.

[23] David Michéa and Dimitri Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. *Geophysical Journal International*, 182(1):389–402, 2010.

[24] Jack Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[25] Antoine P. Petitet. High Performance Computing Linpack Benchmark, HPL - 1.0a. University of Tennessee, Knoxville. Innovative Computing Laboratories. (C) 2000 - 2004.

[26] Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. In David R. Kaeli and Miriam Leeser, editors, *GPGPU*, volume 383 of *ACM International Conference Proceeding Series*, pages 46–51. ACM, 2009.

[27] Andrei Radulescu and Arjan J. C. van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. In *Heterogeneous Computing Workshop*, pages 229–238, 2000.

[28] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *ICCD*, pages 349–356. IEEE Computer Society, 2013.

[29] Hongpeng Huo, Chongchong Sheng, Xinming Hu, Baifeng Wu. An Energy Efficient Task Scheduling Scheme for Heterogeneous GPU-Enhanced Clusters. In ICSAI, pages 623-6277. IEEE Computer Society, 2012.

[30] Keliang Zhang and Baifeng Wu. Task Scheduling Greedy Heuristics for GPU Heterogeneous Cluster Involving the Weights of the Processor. In *IPDPS Workshops*, pages 1817–1827. IEEE, 2013.

[31] Alina Sîrbu and Özalp Babaoglu. Power Consumption Modeling and Prediction in a Hybrid CPU-GPU-MIC Supercomputer (preliminary version). *CoRR*, abs/1601.05961, 2016.

Task-based programming on low-power Nvidia Jetson TK1 manycore architecture: Application to earthquake modeling

Víctor Martínez[1], John García[2], Carlos Barrios[2],
Fabrice Dupros[3], Hideo Aochi[3], and Philippe Navaux[1]

[1] Informatics Institute, Federal University of Rio Grande do Sul (UFRGS),
Av. Bento Gonçalves, 9500, Campus do Vale, 91501-970, Porto Alegre, Brazil
{victor.martinez,navaux}@inf.ufrgs.br
[2] High Performance and Scientific Computing Centre,
Industrial University of Santander, Bucaramanga, Colombia
john.garcia1@correo.uis.edu.co,cbarrios@uis.edu.co
[3] BRGM, 3 Av. Claude Guillemin, Orléans, France
{f.dupros,h.aochi}@brgm.fr

Abstract Simulation of seismic wave propagation is a crucial tool in geophysics for efficient strong motion analysis and risk mitigation. Being particularly CPU-consuming, this problem requieres large-scale computing resources. Although we observe an regular increase of the computing power available, their energy efficiency is a major concern. In this paper, we analyze the use of a low-power manycore architecture, the NVIDIA Jetson TK1 board. We consider a task-based implementation of the finite-differences discretization of the elastodynamics equation method and we evaluate the energy efficiency on various platforms. Considering energy-to-solution metrics, NVIDIA Jetson platform provides significant improvement in comparison with standard hybrid CPU+GPU platforms.

Keywords: Manycore Architectures; task-based programming; GPU accelerators; seismic modeling; energy efficiency; low-power processors

1 Introduction

Scientific applications require huge amount of data and computing power. A wide range of three-dimensional physical problems could not be tackled without large scale simulations. In this case, High Performance Computing is the main solution to achieve robust and accurate solutions. Simulations of seismic wave propagation are critical both for earthquakes analysis and for oil and gas industry (seismic imaging).

From 1993, HPC systems have been built taking into account only peak performance (Flops, floating point operations by second) corresponding to the Top500 list [17]. The situation has changed few years ago, since 2007 HPC systems are also ranked on green500 list according to their energy efficiency (Flops/W), quantity of Flops that can be processed by Watt consumed [8]. The

most powerful machine on top500 (Thianhe-2) achieves a theoretical peak of 54,902.4 TFlops and consumes 17,808.00 kW, then his energy efficiency is only 1,901.54 Mflop/W; i. e. 64th place of green500 list. If next frontier of HPC systems is to obtain a peak of Exaflops with a maximum power consumption of 20MW (50 GFlops/W) this performance cannot be achieved with nowadays technology, because the most energy-efficient machine is the L-CSC with approximately 5 GFlops/W.

In order to overcome this power wall situation, HPC vendors have introduced heterogeneous platforms as a sum of different processors/cores. These architectures have been supported by Graphics Processing Units (GPUs) from NVIDIA, Cell Broadband Engine Architecture (CBEA) from Sony Computer Entertainment, Toshiba and IBM, and Field Programmable Gate Arrays (FPGAs) accelerators solutions from Maxeler Max Nodes and SGI systems, and recently the Xeon Phi Processor from Intel [16].

On the other hand, efficient exploitation of these heterogeneous platforms remains a challenge at the application level. Igual *et. al.* [9] uses a Multi-core Digital Signal Processor (DSP) to solve problems related to General Matrix-Matrix multiplication (GEMM), a common problem in HPC applications. Padoin *et al.* [13] evaluate energy consumption of a unconventional cluster based on Panda boards each one featuring two ARM Cortex A9 cores. In [15] analyzes HPC applications running on low-power embedded platforms (Snowball board) based on ARM processors with an integrated Mali 400 GPU (Mont-Blanc Project). Castro *et al.* [4] analyses energy efficient for seismic model on a low-power manycore processor (MPPA-256).

The Jetson TK1 board [11,12] has been used to solve problems related to Synthetic Aperture Radar (SAR) imaging[6] and a localization system for autonomous mobile robot[14] based on Monte-Carlo problem. Some contributions have been made to improve seismic wave modeling. In [2], the authors implement a seismic model using task-based programming but they don't use manycore architectures for the simulations. Calandra *et al.* [3] evaluate execution of a finite-differences stencil on CPUs, APUs and GPUs but they don't exploit heterogeneous cores perfomance, neither parallel tasks programming.

Efforts at the application level needs to be enhanced by robust runtime systems able to tackle the complexity of heterogeneous platforms in terms of data transfer or tasks scheduling. Last few years, significant efforts have been devoted to provide such tools. For instance, G-Charm[20] is a framework for execution of message-driven parallel applications on hybrid systems, based on Charm++.

In [7] is presented the *XKaapi* for data-flow task programming on heterogeneous architectures, which supports a data-flow task model, but it uses only one locality-aware work stealing scheduler. *StarPU* is a simple tasking API that provides numerical kernel designers with a convenient way to execute parallel tasks over heterogeneous hardware on the one hand, and easily develop and tune powerful scheduling algorithms on the other hand. StarPU is based on the integration of the data-management facility with a task execution engine[1].

Our contribution is to demonstrate that low-power manycore architectures can be used for seismic wave simulations with significant improvement of the energy consumption. The paper proceeds as follows. Section 2 discusses the fundamentals of seismic wave simulation. Then, section 3 presents the Jetson TK1 Board by NVIDIA and section 4 explains the methodology of experiments. Section 5 discusses the performance of simulations based on power consumption and energy efficiency. Finally, section 6 concludes this paper.

2 Earthquake Modeling

2.1 Elastodynamic Equation

Evaluation of damages occurred during strong ground motion is critical for urban planning. This class of problem can be described by the seismic wave equation:

$$\rho \frac{\partial v_i}{\partial t} = \frac{\partial \sigma_{ij}}{\partial j} + F_i \tag{1}$$

Additionally, the constitutive relation in the case of an isotropic medium is:

$$\frac{\partial \sigma_{ij}}{\partial t} = \lambda \delta_{ij} \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right) + \mu \left(\frac{\partial v_i}{\partial j} + \frac{\partial v_j}{\partial i} \right) \tag{2}$$

Where indices i, j, k represent a component of a vector or tensor field in cartesian coordinates (x, y, z), v_i and σ_{ij} represent the velocity and stress field respectively, and F_i denotes an external source force. ρ is the material density and λ and μ are the elastic coefficients known as Lamé parameters. A time derivative is denoted by $\frac{\partial}{\partial t}$ and a spatial derivative with respect to the i-th direction is represented by $\frac{\partial}{\partial i}$. The Kronecker symbol δ_{ij} is equal to 1 if $i = j$ and zero otherwise. Exponents i, j, k indicate the spatial direction with $\sigma^{ijk} = \sigma(i\Delta s, j\Delta s, k\Delta s)$, Δs corresponds to the space step and Δt to the time step.

In order to solve this equation, the standard finite difference method is used. This approach remains very popular due to its implementation simplicity for a wide range of applications [21]. Considering the classical 4-th order in space and second-order in time approximation, the stencil applied for the computation of the velocity component in the x-direction is given by:

$$\begin{aligned} v_x^{(i+\frac{1}{2})jk}\left(l+\tfrac{1}{2}\right) = v_x^{(i+\frac{1}{2})jk}\left(l-\tfrac{1}{2}\right) + a_1 F_x^{(i+\frac{1}{2})jk} \\ + a_2\Big[\frac{\sigma_{xx}^{(i+1)jk}-\sigma_{xx}^{ijk}}{\Delta x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k}-\sigma_{xy}^{(i+\frac{1}{2})(j-\frac{1}{2})k}}{\Delta y} \\ + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})}-\sigma_{xz}^{(i+\frac{1}{2})j(k-\frac{1}{2})}}{\Delta z}\Big] \\ - a_3\Big[\frac{\sigma_{xx}^{(i+2)jk}-\sigma_{xx}^{(i-1)jk}}{\Delta x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{3}{2})k}-\sigma_{xy}^{(i+\frac{1}{2})(j-\frac{3}{2})k}}{\Delta y} \\ + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{3}{2})}-\sigma_{xz}^{(i+\frac{1}{2})j(k-\frac{3}{2})}}{\Delta z}\Big] \end{aligned} \tag{3}$$

The same numerical scheme is used to compute the stress components.

2.2 Ondes3D on Top of StarPU Runtime System

The previous equations are solved in the Ondes3D software package developed at BRGM (French Geological survey). Ondes3D uses the finite-differences method on a regular grid. It behaves like a stencil code and naturally express data parallelism whereas StarPU works on a task parallelism level [1]. This runtime system allows us to decompose the original problem into several tasks (computation or communications) creating a Directed Acyclic Graph (DAG) that will be smoothly scheduled on the heterogeneous cores of the platform.

To express task parallelism in Ondes3D, the tip is to split the model into a fine grained blocks grid, each part of the code executed on a single block being a task. This has been already done in the examples which come along with StarPU. One of them is called stencil and it implements the game of live to demonstrate how to use StarPU with a stencil based code.

To save time and benefit from the experience of the developers of StarPU, we used this example as a basement to port Ondes3D on top of StarPU. The overall algorithmic is described in figure 1. We reuse the numerical kernels implemented for GPU and CPU architecture. A detailed description of these implementations could be found in [5,10].

```
1: StarPU Initialization
2: procedure Initialization of data structures
3:     Read_parameters(file)
4:     Model_slicing(Create blocks)
5: end procedure
6: procedure Create all tasks
7:     Create dummy start task(Time measurement)
8:     for each iteration do
9:         for each block do
10:            Create task(update source)
11:            Create task(compute velocity)
12:            Create task(save boundaries for velocity)
13:            Create task(compute stress)
14:            Create task(save boundaries for stress)
15:            Create task(record seismograms)
16:        end for
17:        Create dummy end task(Synchronization)
18:    end for
19:    Release dummy start task
20:    Wait(end of the dummy end task)
21: end procedure
```

Figure 1. Ondes3D algorithm on top of StarPU

From algorithm, we have defined two kind of tasks for each block solving, for each timestep iteration: communication tasks (update source, save boundaries and record) for thread synchronization and data sharing between heterogenous cores, and computation tasks (velocity and stress) to solve the finite-differences method described in equation 3.

3 Nvidia Jetson TK1 Board

Nvidia Jetson board is a low-power manycore architecture that uses a Tegra K1 mobile processor developed by NVIDIA to have the same advanced features & architecture as a modern desktop GPU, while still using the low power draw of a mobile chip. Therefore Tegra K1 allows embedded devices to use the exact same CUDA code that would also run on a desktop GPU. The Tegra K1 is a System-on-a-Chip (SoC) that combines a Kepler GPU architecture, one of the most recent, with a ARM-based processor, a low-power CPU, to obtain computing capabilities and breakthrough power efficiency. The NVIDIA Tegra K1 mobile processor is designed to exploit capabilities of mobile CPU processor (delivers energy efficient implementations) and advanced GPU-accelerated computing capabilities.

Some of the key features of the Tegra K1 SoC (System-on-a-Chip) architecture are described in [11,12] and shown in figure 2. For our purposes, processors and memory on this board are:

- CPU: ARM based processor, 4-PLUS-1 @2.32GHz Cortex A15 "r3" CPU battery-saving shadow-core, an architecture that delivers higher performance and is more power efficient than the previous generation.
- GPU: Kepler GPU architecture, "GK20a" utilizes 192 SM3.2 CUDA cores, GPU computing with NVIDIA CUDA 6 support, breakthrough power efficiency and performance for GPU-accelerated computing applications.
- DRAM: shared memory, 2GB DDR3L 933MHz EMC x16 using 64-bit data width.

According to wiki resource in [22], *typical power consumption* is between 1 to 5 Watts. The Tegra K1 SOC in Jetson TK1 is aimed to use between 0.6W to 3W of power during normal use and rarely uses more than 4W, but is able to reach 15W if it managed to push the CPU, GPU, camera ISP's and codec hardware. Meanwhile, the rest of the Jetson TK1 board uses between 1.5W to 45W depending on what it's turned on and plugged into it (through USB, mini-PCIe, SATA, SD-card, HDMI, audio, GPIO, expansion port, etc). The absolute max power draw of Jetson TK1 if it pushed everything to the limit and use every port including SATA and PCIe is 58W.

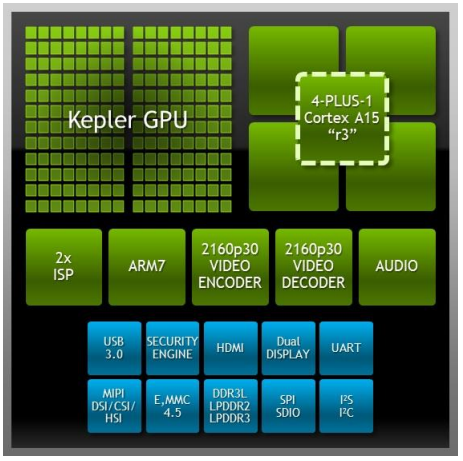


Figure 2. Architecture of Tegra TK1

Jetson TK1 comes pre-installed with Linux4 Tegra OS, Ubuntu 14.04.2 LTS (GNU/Linux 3.10.24-g6a2d13a armv7l) [22]. It is remarkable that RAM is shared between CPU and GPU cores, without PCI-Express transferences, then this board uses only unified memory management in CUDA 6.

4 Experimental Setup

4.1 Methodology

Our objective is to demonstrate that Nvidia Jetson manycore architecture represents an alternative for energy efficient seismic wave modeling. We use a simulation scenario for the three-dimensional model that includes a Cartesian mesh of 7.2 million grid points (300×300×80). The memory consumption for this problem is 625 MB of memory.

Based on StarPU task-based implementation, two computing kernels (velocity and stress components) must be scheduled. We need to use 36 parallel computing tasks and 144 parallel communication tasks (data sharing between blocks).

StarPU runtime system has two flags for heterogenous cores utilization (workers): STARPU_NCPU to define number of CPU cores and STARPU_NCUDA to define number of GPU cards to be used on computation of kernels, we use 3 cores for CPU computation on each architecture and 1 core for GPU management. Also we used STARPU_CUDA_ASYNC to execute CUDA kernels in a concurrent way, if available.

We selected DMDAR (Deque Model Data Aware Ready) scheduler that maps tasks onto workers using an history-based kernel performance mode such that per-worker task queues are sorted according to the number of already available dependent pieces of data. To make this, we worked with 1.2 version of StarPU.

We use three metrics to discuss the performance obtained :

- time-to-solution (execution time for the simulation)
- energy consumption (using platform sensors by NVPROF).
- energy efficiency (FLOPS/Watt)

4.2 Description of the Heterogeneous Platforms

Three parallel platforms has been used to compare the performance of our task-based implementation. We consider the Nvidia Jetson Board and two standard hybrid nodes in order to measure the parallel performance and the energy consumption:

1. **Server**: The sever machine supports 16 nodes, each one with 8 GPUs [18]. We use one node with the following characteristics: two Intel® Xeon® E5645 processors @2.40GHz (2×6 physical cores) and eight accelerators NVIDIA® Tesla™ M2075 with 4.9 GB of RAM (8×448 CUDA cores).
2. **Desktop**: This platform can be viewed as commodity-based architecture. For this machine we have: one processor Intel® Core™ i7-930 @2.80GHz. (4 physical cores) and one accelerator NVIDIA® Tesla™ K20c with 5 GB of RAM (2496 CUDA cores).
3. **Jetson**: We used the NVIDIA® Jetson TK1 development kit with NVIDIA 4-Plus-1™, quad-core ARM® Cortex-A15 CPU and 2 GB of shared memory for CPU and GPU cores. We turned off HDMI and USB ports.

5 Experimental Results

5.1 Time Results

The results described in Figure 3 confirm the assumption that Nvidia Jetson board exhibits poor performance considering the time-to-solution metrics. We observe that the HPC node is 2.05x faster than the Nvidia Jetson board. For the commodity-based architecture, the ratio is 1.59x. This is mainly coming from the different levels of performance of the GPU available on each platform.

If we consider ratio between computation time (tasks to solve velocity and stress kernels) and communication time (tasks to transfer data between global RAM and GPU RAM, and thread synchronization) then we found that Jetson board is processing 66.40% of time, while desktop and server are processing only 29.39% and 53.11% of time respectively. This is because Jetson has a shared memory between CPU cores and GPU. As a result, we only pay the costs of threads synchronization of GPU kernels and CPU functions. Due to the size of the test-case (625 MB) and the speedup ratio between the GPU and the CPU cores, StarPU uses both CPU and GPU cores to schedule the computing tasks on the Nvidia Jetson board. For the desktop machine and the server node the runtime system only schedules the computation tasks on the GPU and the remaining communication tasks on the CPU cores.

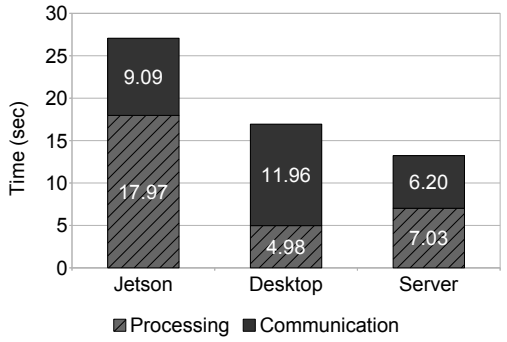


Figure 3. Comparison of Time-to-solution metrics for Nvidia Jetson, HPC node and desktop platform.

5.2 Energy results

In this section, we analyze the energy consumption. For Jetson board we assume that power consumption corresponds to 4W, because we can not measure it, this value represents an upper-bound when only CPU and GPU cores are working [12,22], and we disabled interface ports (USB, HDMI); for desktop and server machines we got power measures from Nvidia Profiler. For the computation phase, we have measured 83.40W and 123.49W for the desktop machine and the server node respectively.

If we compare energy consumption presented on figure 4 (left), Jetson board reduces the energy consumption by a factor 12.21x in comparison with the server node and by a factor 2.08x with respect to the desktop machine. Obviously, the desktop machine is tuned for energy-efficiency contrary to high-end HPC node.

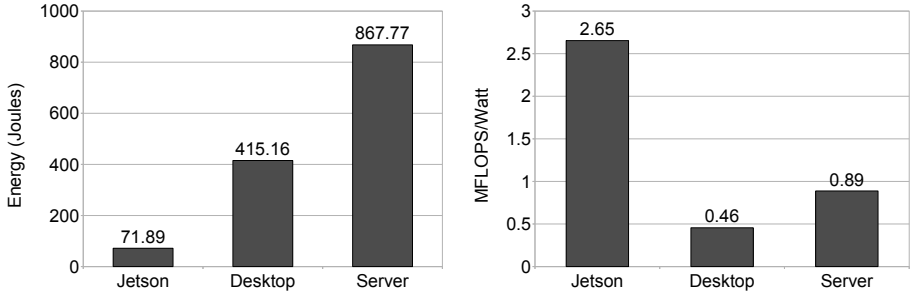


Figure 4. Energy consumption (left) and energy efficiency (right) for the earthquake modeling on three heterogenous architectures.

Finally, we compute the flops/watt ratio for each platform (figure 4, right). We found that Jetson board is 5.82x more efficient than desktop machine and 2.99x than a standard HPC node. One important remarks should be underlined at this stage. The problem under study is tailored to fit in the memory available on the Nvidia Jetson Card (2 GB). This parameter is of great importance to discuss the overall performance as the size of the problem (and the number of blocks) could significantly influence the speedup on GPU architectures [10]. A complete study of these parameters is out of the scope of this p6aper.

6 Conclusion

In this paper, we demonstrate that a low-power manycore architectures can be a alternative if we want to solve seismic modeling with a better energy efficiency. Our solution makes a better usage of the available resources (CPU and GPU cores) with a significant reduction of the communication cost (33.60%). We also underline good results in terms of energy-to-solution compared to common HPC systems. The size of problems that could be tackled is limited by the amount of memory available on Jetson board (2GB).

We are currently working on new strategies in order to use an unconventional cluster of Jetson boards. We plan to add more boards to solve the simulation and use techniques of distributed memory like MPI, that can be easily integrated with StarPU.

Acknowledgments

This work have been granted by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), the HPC-GA project funded by the FP7-PEOPLE under grant agreement number 295217 and BRGM Carnot-institute. Experiments presented in this paper were carried out using the GUANE-1 experimental testbed, being developed under the Universidad Industrial de Santander (SC3UIS) High Performance and Scientific Computing Centre, development action with support from UIS Vicerrectoría de Investigación y Extensión (VIE-UIS) and several UIS research groups as well as other funding bodies (see http://grid.uis.edu.co and http://www.sc3.uis.edu.co). This research was accomplished in the context of the International Joint Laboratory LICIA.

References

1. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Concurrency and Computation: Practice and Experience. 23, pp. 187–198 (2011)
2. Boillot, L., Bosilca, G., Agullo, E., Calandra, H.: Task-Based Programming for Seismic Imaging: Preliminary Results. In: IEEE 6th Intl Symp on Cyberspace

Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on High Performance Computing and Communications, pp. 1259–1266. IEEE Press, New York (2014)
3. Calandra, H., Dolbeau, R., Fortin, P., Lamotte, J.-L., Said, I.: Evaluation of Successive CPUs/APUs/GPUs Based on an OpenCL Finite Difference Stencil. In: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 405-409. IEEE Press, New York (2013)
4. Castro, M., Dupros, F., Francesquini, E., Mehaut, J.-F., Navaux, P.O.A.: Energy Efficient Seismic Wave Propagation Simulation on a Low-Power Manycore Processor. In: IEEE 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 57–64. IEEE Press, New York (2014)
5. Dupros, F., Do, H., Aochi, H.: On scalability issues of the elastodynamics equations on multicore platforms. In Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013, ser. Procedia Computer Science, vol. 18. Elsevier, 2013, pp. 1226–1234.
6. Fatica, M:, Phillips, E.: Synthetic Aperture Radar imaging on a CUDA-enabled mobile platform. In: 2014 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–5. IEEE Press, New York (2014)
7. Gautier, T., Lima, J. V. F., Maillard, N., Raffin, B.: XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In: IEEE 27th International Symposium on Parallel and Distributed Processing, pp.1299–1308. IEEE Computer Society, Washington, DC, USA (2013)
8. Green500 list `http://www.green500.org`
9. Igual, F.D., Ali, M., Friedmann, A., Stotzer, E., Wentz, T., Van De Geijn, R.A.: Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC. In: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1-11. IEEE Press, New York (2012)
10. Michéa, D., Dimitri Komatitsc, D.: Accelerating a 3D finite-difference wave propagation code using GPU graphics cards. Geophysical Journal International. 182, 389–402 (2010)
11. NVIDIA: Technical Brief, NVIDIA Jetson TK1 Development Kit (Bringing GPU-accelerated computing to Embedded Systems). 2014. `http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf`
12. NVIDIA: Jetson TK1 Development Kit (Specification). 2014. `http://developer.download.nvidia.com/embedded/jetson/TK1/docs/3_HWDesignDev/JTK1_DevKit_Specification.pdf`
13. Padoin, E. L., Oliveira, D. A. G., Velho, P., Navaux, P. O. A.: Evaluating Performance and Energy on ARM-based Clusters for High Performance Computing. In: 2012 41st International Conference on Parallel Processing Workshops, pp. 165–172. IEEE Press, New York (2012)
14. Rud, M. N., Pantiykchin, A. R.: Development of GPU-accelerated localization system for autonomous mobile robot. In: 2014 International Conference on Mechanical Engineering, Automation and Control Systems (MEACS), pp. 16–18. IEEE Press, New York (2014)
15. Stanisic, L., Videau, B., Cronsioe, J., Degomme, A., Marangozova-Martin, V., Legrand, A., Méhaut, J.-F.: Performance Analysis of HPC Applications on Low-Power Embedded Platforms. In: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 475–480. IEEE Press, New York (2013)
16. Stojanovic, S., Bojic, D., Bojovic, M., Valero, M., Milutinovic, V.: An overview of selected hybrid and reconfigurable architectures. In: 2012 IEEE International

Conference on Industrial Technology (ICIT), pp. 444–449. IEEE Press, New York (2012)
17. Top500 list. http://www.top500.list
18. Universidad Industrial de Santander: Super Computación y Cálculo Científico UIS, http://www.sc3.uis.edu.co/servicios/hardware/
19. Université de Bordeaux, CNRS, INRIA: StarPU Handbook, http://starpu.gforge.inria.fr/doc/starpu.pdf
20. Vasudevan, R., Vadhiyar, S. S. and Kalé, L. V.: G-Charm: An Adaptive Runtime System for Message-driven Parallel Applications on Hybrid Systems. In: 27th International ACM Conference on International Conference on Supercomputing, pp. 349–358. ACM, New York, NY (2013)
21. Virieux, J.: P-SV wave propagation in heterogeneous media; velocity-stress finite-difference method. Geophysics. 51, 889–901(1986)
22. Wiki for embedded Tegra & the Jetson TK1 board. http://elinux.org/Jetson_TK1

eGPU for Monitoring Performance and Power Consumption on Multi-GPUs

John A. G. Henao[1,2], Víctor M. Abaunza[2], Philippe O. A. Navaux[2], Carlos J. B. Hernández[1]

[1]High Performance and Scientific Computing Center, University Industrial of Santander, BGA-Colombia
[2]Parallel and Distributed Processing Group, Informatics Institute, Federal University of Rio Grande do Sul, POA-Brazil
[1,2]john.garcia1@correo.uis.edu.co, [2]victor.martinez@inf.ufrgs.br, [2]navaux@inf.ufrgs.br, [1]cbarrios@uis.edu.co

Abstract

The evaluation of performance and power consumption is a key step in the design of applications for large computing systems, such as supercomputers, clusters with nodes that have manycores and multi-GPUs. Researchers must design several experiments for workload characterization by observing the architectural implications of different combinations of parameters, such as problem size, number of cores per GPUs, number of process MPI, and observe the resulting clock frequency, memory usage, bandwidth, and power consumption, factors which determine the performance and energy efficiency of their workload implementation. The major problem in performance evaluation consists on the design space exploration given so many parameters. If the resulting data is not properly collected and the parameters of evaluation performance are not well organized, the design exploration may lead to wrong conclusions. This paper presents eGPU such as monitoring tool that can be used on evaluations of performance with many experiments which aim measure and understand the behavior of factors that determine the energy efficiency in nodes with multi-GPUS. eGPU is a monitor to centralize and automate the data captured in runtime while is executed in parallel with the Linpack Benchmark and displays information via sequenceplots, statistical tables, bar graph and shows results in terms of energy efficiency.

1. Introduction

The power consumption is becoming the major concern for Exascale Systems. *"Indeed, a key observation from the study from DARPA (Technology Challenges in Achieving Exascale Systems) is that it may be easier to solve the power problem associated with base computation than it will be to reduce the problem of transporting data from one site to another on the same chip, between closely coupled chips in a common package, or between different racks on opposite sides of a large machine room, or on storing data in the aggregate memory hierarchy"* [1]. In this order, the evaluation of performance and power measurement need to analyze multiple tests under different combinations of parameters to observe the key factors that determine the energy efficiency in terms of energy per computation, energy per data transport, energy per memory access, or energy per secondary storage unit of a workload model in computing system extrapolation of science problems of today to Exascale.

The workload widely used by the Green500 and the Top500 in evaluation of performance and power measurement evaluation of supercomputers is the LINPACK Benchmark (HPL) [2]. The Linpack is an algorithm to solve a dense system of linear equations, where is allowed increase the problem size to find the performance numbers that reflect the largest problem can be run on a supercomputer [3]. The goal of this paper is to present the utilization of the first version of eGPU for automate the data collection that includes clock frequency, memory usage, and power consumption across the sensors of the NVIDIA GPU.

2. eGPU Monitor Structure

eGPU is a type of batch monitor that is formed by two levels [4]. One level to capture data in runtime, composed by three events, and a separate level that can be later used to visualize data online, composed by four events, such as shown in the Figure 1.

The first level is used to run a script called $eGPUrecord.sh$, where the execution steps and exported libraries needed to run Linpack Benchmark are declared, such as BLAS Library and many oth-

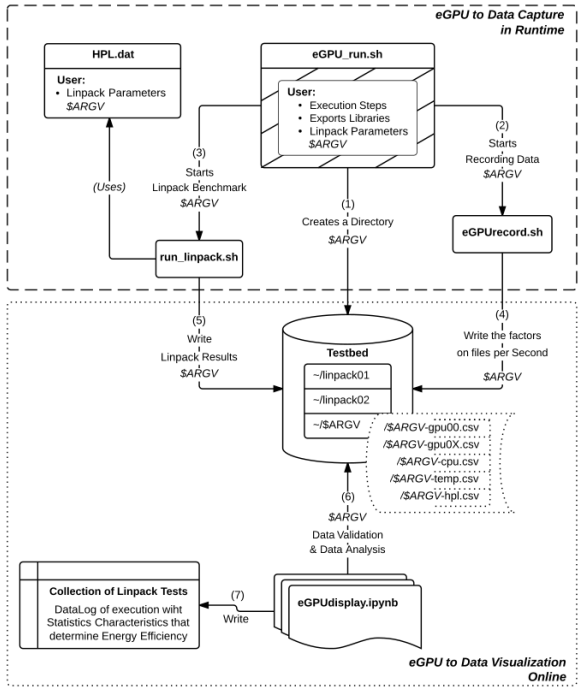


Figure 1. eGPU Monitor Structure.

ers: $(-lcuda - lcudart - lcublas - lmkl\_intel\_lp64 - lmkl\_intel\_thread - lmkl\_core)$. The user writes the same Linpack parameters defined in the file $HPL.dat$ to execute the experiment. Those parameters are used as a unique key called $ARGV$ to share the argument in all the following events that uniquely identify each test of Linpack benchmark. The event (1) creates a directory in a repository that we called $testbet$, where the data is centralized. The events (2) and (4) are used to run a script called $eGPUrecord.sh$ with special tasks written in AWK, C and environment variables, for text processing and data collection in the $testbet$ repository. $eGPUrecord.sh$ makes the data extraction and writes: one separate file for each GPU with the data (Streaming Multiprocessor Clock Frequency -SM-, Memory Clock Frequency, Memory Usage, Power Consumption), these all have new data registered every second. The events (3) and (5) start the script $run\_linpack$ to solve a dense system of linear equations, calculate performance obtained and write results when it finishes.

The second level is used at post-processing for data visualization. It displays information via sequence plots, statistical tables, histograms and shows results in terms of energy efficiency. The event (6) displays information of tests through the IPython Notebook Viewer in a program called $eGPUdisplay.ipynb$ that contains predefined code routines written in Python, where the researchers have to type the argument $ARGV$ to identify the unique test of the Linpack benchmark. eGPU will then shows the sequence plots for the characteristics of each GPU in each node and the following functions. At this point, it will also make a data validation with arguments received from the events (2) and (3) to shows statistical tables with the mean and standard deviation for all the data of each GPU of each node. Subsequently eGPU displays the Linpack results with time spent, performance, power consumption and energy efficiency. Finally eGPU displays a bar graph with a comparison between the node power consumption when idle and when running the Linpack Benchmark.

3. Experimental Procedures and Results

This section presents the use of eGPU to capture data and visualize it in one test with Linpack benchmark. The data were captured in the evaluation of the influence of the number of GPUs and processes in the performance and energy efficiency on nodes of the GUANE cluster.[1] The computational resources used for this experiment are part of one node of the 'A' settings of cluster GUANE that consists in ProLiant SL390s-G7 computing nodes such as shown in Table 1.

Setting GUANE	A	B	C
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Processor by node (#)	2	2	2
Clock frequency (GHz)	2.40	2.40	2.670
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
GPUs by node (#)	8	8	8
Memory DDR (GB)	104	104	104
SAS disk (GB)	200	200	200
Gigabit Ethernet (Gbps)	10	10	10
InfiniBand (IB)	1	1	1

Table 1. Settings GUANE Cluster Nodes.

1 GUANE is a heterogeneous cluster in the High Performance and Scientific Computing Center -SC3UIS, located at technological camp of Industrial University of Santander at Piedecuesta, a municipality of the metropolitan area of Bucaramanga in Santander, Colombia.

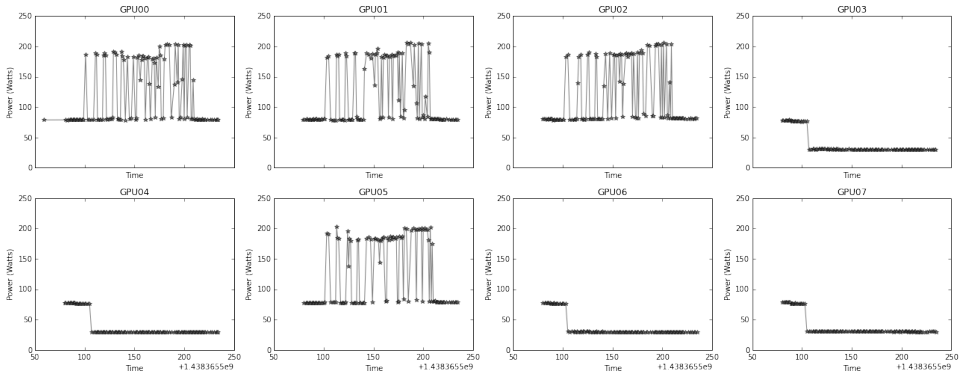


Figure 2. eGPU-Sequenceplot to analyzes of Power Consumption by each GPU.

In this experimental procedure we used HPL.2.0 version configured for Tesla GPUs [5]. The Linpack parameters used are shown in Table 2. They were written in the file $HPL.dat$ and the script $eGPU\_run.sh$ to perform the test for the Linpack benchmark. Then eGPU starts to do text processing and centralize the data collection in the repository $testbed$, and links all events of $\$ARGV$, which represents a unique structure with the parameters used for this test. The script $run\_linpack$ writes performance results in the repository $testbet$ when it is finished.

Matrix size	49152
Block size	1024
GPU Used	4
Cores per GPU	3
Process MPI	4

Table 2. The Linpack parameters used in experimental procedure.

After having captured data, an iterative session is created via $tunneling\ ssh$ to execute a remote IPython Notebook Viewer of the second level of eGPU, in order to visualize data. Once the program $eGPUdisplay.ipynb$ is initialized, we must provide the path with the argument $\$ARGV$ for this experimental procedure. Below we show eight sequence plots with the data (SM Clock Frequency, Memory Clock Frequency, Memory Usage, Power Draw) for each GPU of one node. In the Figure 2, we show the sequence plots for Power Consumption of each GPU, enabling us to observe which GPUs worked while Linpack executed. Next, eGPU shows the statistics to analyze the data that determines energy efficiency such as shown in Table 3. This experiment procedure shows the 4 GPUs that are inactive down to average frequencies of 212MHz for SM clock and 340MHz for memory clock, in order to reduce power consumption. The other 4 GPUs actually working have constant frequencies of 1147MHz for SM clock and 1566MHz for memory clock, with a constant memory usage of 2128MiB, thus generating an average power of 120 watts between working GPUs. The standard deviation of 51 watts shows that execution can be improved, for instance, if we increase the number of cores per GPU or the number of MPI processes to make full utilization of GPUs cores.

Subsequently eGPU displays the amount of energy used by each GPU between $idlepower$ and $runpower$, in terms of Joules. Idle power is the power consumption when a GPU is on but no application is running, for which we found an average power consumption of 78.82 Watts for all GPUs in the node. In comparison, when the GPUs are running, the average power consumption is 78.7 Watts, since the 4 GPUs that are inactive scale down the SM and memory frequencies, such as shown in Figure 3.

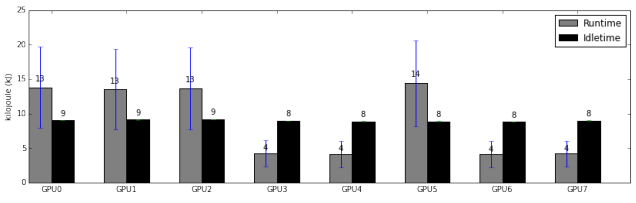


Figure 3. eGPU-Bar graph to analysis of energy used between Idletime and Runtime by each GPU.

Factors	GPU00	GPU01	GPU02	GPU03	GPU04	GPU05	GPU06	GPU007
SM(MHz) Mean	1147	1147	1147	215.74	223.73	1147	204.66	206.65
SM(MHz) Std	0	0	0	282.46	301.54	0	262.64	268.84
Memory(MHz) Mean	1566	1566	1566	346.53	346.53	1566	334.09	334.09
Memory(MHz) Std	0	0	0	507.90	507.90	0	495.24	495.24
MEM Usage(MiB) Mean	2128.9	2128.89	2128.9	10	10	2128.87	10	10
1566 MEM Usage(MiB) Std	0.30	0.31	0.30	0	0	0.33	0	0
Power(Watts) Mean	120.55	118.01	119.12	37.22	36.27	125.65	36.07	36.76
Power(Watts) Std	51.29	50.92	51.36	16.62	16.78	54.32	16.26	16.00
Energy(KJ) Mean	13.80	13.51	13.64	4.26	4.15	14.39	4.13	4.21
Energy(KJ) Std	5.87	5.83	5.88	1.90	1.92	6.22	1.86	1.83

Table 3. eGPU-Statistics to analyzes the factors that determine Energy Efficient.

Table 4 shows the results obtained for performance, where eGPU observed 691.20 GFLOPS and time spent solving the Linpack, and calculated a latency between the time you start to capture data and the time eGPU start Linpack. When using 4 GPUs, with 3 cores per GPU and 4 processes MPI to solve a 49152∗49152 system of linear equations, one node of Cluster GUANE achieved the energy efficiency of 1097.68 MFLOPS/W.

eGPUrecord Time:	121 sec
eGPUrecord Latency:	6 sec
HPL2.0 Time:	114.54 sec
HPL2.0 Performance:	691.20 GFLOPS
Power Consumption:	629.65 Watts
Energy Consumption:	72.12 kJ
Energy Efficiency:	1097.68 MFLOPS/W

Table 4. eGPU-Results to analyzes of Linpack benchmark in this experimental procedure.

4. Conclusions

We constructed eGPU to facilitate the collection and visualization of data to analyze many tests under different combinations of parameters and observe the granularity of the factors that determine energy efficiency in clusters with multi-GPUs. The method we use is focused ony analyzing previously compiled applications, where researchers do not need to orchestrate the code to execute eGPU, ensuring the integrity of the results. Based on the experiment procedures and results presented, eGPU is a good alternative to analyze power consumption in clusters with multi-GPUs from a software level, and can be complemented with other energy monitors that are designed to be plugged-in directly into the power supply to make holistic measures in clusters with multi-GPUs.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. Williams, and K. Yelick. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead, 2008.

[2] Run Rules, Green500. Energy Efficient High Performance Computing Power Measurement Methodology. Version: 1.2RC2, 2014.

[3] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Electrical Engineering and Computer Science Department, University of Tennessee (TN 37996-1301), Computer Science and Mathematics Division, Oak Ridge National Laboratory (TN 37831), University of Manchester, 2014.

[4] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[5] Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. In David R. Kaeli and Miriam Leeser, editors, *GPGPU*, volume 383 of *ACM International Conference Proceeding Series*, pages 46–51. ACM, 2009.

enerGyPU and enerGyPhi Monitor for Power Consumption and Performance Evaluation on Nvidia Tesla GPU and Intel Xeon Phi

John A. García H.[1,2], Esteban Hernandez B.[1,3], Carlos E. Montenegro[3], Philippe O. Navaux[2], Carlos J. Barrios H.[1]

[1]High Performance and Scientific Computing Center - SC3, Universidad Industrial de Santander - UIS
[2]Parallel and Distributed Processing Group - GPPD, Universidade Federal do Rio Grande do Sul - UFRGS
[3]Interoperability of Academic Networks Group - GIIRA, Universidad Distrital Francisco José de Caldas - UD
[1,2]john.garcia1@correo.uis.edu.co, {[1,3]ejhernandezb, [3]cemontenegrom}@udistrital.edu.co, [2]navaux@inf.ufrgs.br, [1]cbarrios@uis.edu.co

***Abstract*—The evaluation of performance and power consumption is a key step in the design of applications for large computational systems as supercomputers and clusters (multicore and accelerator nodes, multicore and coprocessor nodes, manycore and accelerator nodes). In these systems the developers must design several experiments for workload characterization observing the architectural implications when using different combinations of computational resources such as number of GPU, number of cores for processing, number of cores for administration of GPU, number of MPI processes and thread affinity policy. It should also engage factors as the clock frequency and memory usage as well select the combination of computational resources that increases the performance and minimizes the power consumption. This research proposes an integrated energy-aware scheme called efficiently energetic acceleration (EEA) for large-scale scientific applications running on heterogeneous architectures. This paper shows the use of a monitoring tool with two components called enerGyPU and enerGyPhi to recording EEA control factors in runtime on two environments: one cluster with multicore and accelerator nodes (2-CPU/8-GPU) and one server with multiple cores and one coprocessor (2-CPU/1-MIC). These monitors allow to analyze multiple testing results under different parameter combinations to observe the EEA control factors that determine the energy efficiency.**

***Index Terms*—Energy efficiency, Energy-aware EEA scheme enerGyPU, enerGyPhi, Power capping technique, Performance evaluation.**

I. Introduction

Heterogeneous parallel programming has two problems on large computation systems: one is the increase of power consumption on supercomputers in proportion to the amount of computational resources used to obtain high performance, and the other problem is the underuse these resources by scientifics applications with a inexact distribution of tasks. Select the optimal computational resources and make a good mapping of task granularity is the main challenge for build the next generation of Exascale Systems [1].

The power consumption increases when the heterogeneous architectures increase the number of cores inside a chip, such as Intel Xeon Phi or Nvidia Tesla GPU. Depending on the circuit design (power gating, clock gating, memory bandwidth usage), the gradient of power consumption also varies [2]. However, mapping of task granularity of an application to achieve high performance with a lower power consumption is even more difficult. The heterogeneous parallel programming required different scheduling algorithms and number of threads to use Simple Instruction Multiple Thread (SIMT) for Nvidia Tesla GPU or Single Instruction Multiple Data (SIMD) for Intel Xeon Phi [3]. On Intel Xeon Phi, the number of Cores, threads per core and level of affinity is a critical aspect to achieve better level of performance and know the factors where can be minimized the power consumption. On Nvidia Tesla GPU, the kernel grid and number of threads/blocks, streaming multiprocessor clock frequency, memory clock frequency, memory usage, bandwidth usage is a key aspect.

This paper shows the use of monitor called enerGyPU and enerGyPhi to centralize and automate the capture of EEA control factors in runtime while is executed in parallel with the scientific application and displays information via sequence plots, statistical tables, bar graphs and shows results in terms of energy efficiency. The aim of this research is analyze the power capping technique on multiGPU with different combinations of computational resources to obtain the GPU power levels and build the GPU power cost function for that the programmer or a load balancing framework can select of computational resources at static time to mapping parallel task granularity. This work uses the Highly-Parallel LINPACK (HPL) Benchmark a general dense matrix problem as case of study for solving systems of linear equations in large scientific applications to evaluate the performance and the power consumption on machines with multicore and accelerator/coprocessor. The HPL is widely used by Top500 and Green500 in the evaluation of performance and power consumption on supercomputers [4].

This paper is organized as follows: the section II presents the related work. The section III shows the energy-aware scheme proposed. The section IV defines the computing resources and selected energy metrics. The section V describes the HPL benchmark as case of study workload. The section VI presents the EEA parameters used by enerGyPU and enerGyPhi. The section VII describes the experimental procedures with power

consumption and performance analysis. Finally we conclude in the section VIII.

II. Related Work

A. Power Capping and Load Balance for CPU and GPU

Power capping is a technique widely used to save power consumption from the supercomputers through Dynamic Voltage and Frequency Scaling (DVFS), this technique allows adjust manual or automatically the frequency and voltage in CPU and recently in the GPU. Komoda et al. [5] uses the power capping to manage device frequencies and task mapping at the beginning of the execution and presents a empirical model with a small number of profiles to predict the execution time and the power consumption using the frequency of CPU and GPU as main parameters. The GreenGPU framework proposes an energy management and load balance in two-tier, in the first tier dynamically splits and distributes workloads to GPU and CPU trying can finish approximately at the same time, and the second tier, GreenGPU dynamically throttles the frequencies of GPU cores and memory in a coordinated manner, based on their utilizations [6]. These schemes of power capping and load balance work well when is used a CPU and a GPU, nevertheless, when having many nodes with multi-GPUs (4, 6 and 8 GPUs) the problem is transferred to know the amount of computational resources that are needed to task mapping for a specific workload. Hogn and Wang [2] proposes an integrated power and performance (IPP) prediction model for a GPU architecture. IPP predicts performance per watt and also the optimal number of cores to achieve energy savings in a GPU. The number of active cores predicted by IPP only involves the number of blocks inside an application, unless they change the hardware or the thread scheduler.

B. Energy-aware Scheme and Power Modeling for Clusters

Huo et al. [7] presents an energy efficient task scheduling scheme for heterogeneous GPU clusters can switch between load balancing and dynamic resource scaling according to the node selection policy based on GPU utilization of the particular task, it can decrease the static energy consumption of GPU in idle status. This scheme uses the evaluation method of suitable GPU core number for particular task based on performance per watt has been proposed by Hogn and Wang [2]. Zhang et al. [8] proposes WSLSA greedly heuristic which can involve the weights of the processor to approximately optimal mapping between tasks and computational resources. They also partition all task to six classification based on the degree of parallelism, workload and input data size of task. The input data is a key parameter for determining the total number of threads and the workload is a key parameter for task sheduling. The WSLSA algorithm may not efficient when the tasks with big workloads are too much according to the task assigning rule, however, it is possible to tune WSLSA algorithm. Sîrbu and Babaoglu [9] present a recent study of power consumption modeling and prediction in a hybrid CPU-GPU-MIC Supercomputer. Using data from a dedicated monitoring framework, they build a data-driven model of power consumption for each user in the system and use it to predict the power requirements of future jobs. They used the *CountVectorizer* class in the *scikit-learn* python package and using the frequencies as main power parameters for predicting power usage per job into 5 regression analyses, one for each component type, and then sum the predicted component powers to obtain an estimate for the global job power.

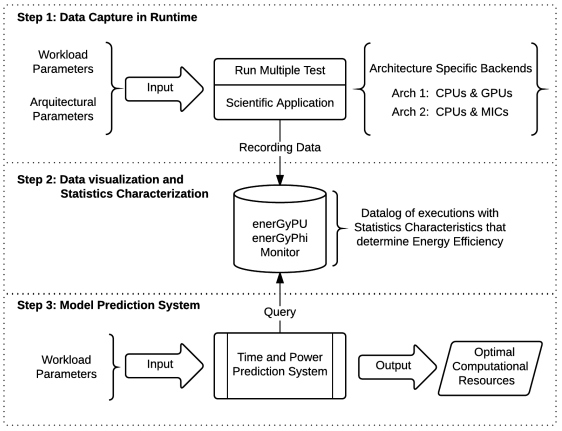


Fig. 1: Efficiently Energetic Acceleration (EEA) Scheme

III. EEA Energy-aware Scheme

This paper proposes an integrated energy-aware scheme called efficiently energetic acceleration (EEA) for scientific applications of large-scale on heterogeneous architectures. The EEA scheme has a workflow of three steps as shows in the Figure 1. The data capture in runtime, in this step is executed the enerGyPU or enerGPhi monitor tool in parallel with the scientific application using different combinations of computational resources applying the power capping technique for nodes with multi-GPUs. In the second step, the data visualization and statistics characterization used a separate level of enerGyPU and enerGPhi monitor tool for analysis the key factors by results of each experiment in terms of energy efficiency and estimated the power levels. Finally using data from enerGyPU and enerGPhi monitor is built the cost functions and model prediction system for obtain the optimal computational resources, at stactic time to mapping parallel task granularity of scientific applications on heterogeneous architectures.

A. Data Capture in Runtime

The first step is very important because is the workload and architectural parameter selection for data capture on environments with CPU, GPU and MIC. Researchers must design full or fractional factorial experiments [10] with different combinations of computational resources and different size as input data for recording of factors, such as Streaming Multiprocessor frequency, GPU memory frequency and usage with diferents number of devices, using the power

capping technique for saving power consumption in data transfer times. Input data and number of devices are key parameters for determining the total number of threads inside a GPU and the kind workload determines the task scheduling.

The task scheduling algorithms uses a directed acyclic graph (DAG), therefore, task mapping in different problems as LU decomposition, laplace equation solver, stencil algorithms and other, varied the task graph granularities, by varying the communication-to-computation ratio. [11]. This work uses the HPL, which is an implementation of the LU decomposition to solve a dense $N * N$ system of linear equations as a case study of one workload. The parameter selection for enerGyPU and enerGPhi monitor is shows in the section VI.

B. Data visualization and statistics characterization

The second step used enerGyPU or enerGPhi at post-processing for data visualization and statistics characterization. It displays information via sequence plots, statistical tables, histograms and shows results in terms of energy efficiency. This displays information of tests through the IPython Notebook Viewer in a program called *enerGyPUdisplay.ipynb* that contains predefined code routines written in Python, where the researchers have to type the argument $ARGV$ to identify the unique test of the Linpack benchmark. The statistics characterization is used for analysis the key factors by results of each experiment in terms of energy efficiency and estimated the power levels.

C. Model prediction System

The three step used data from enerGyPU and enerGPhi monitor is built the cost functions and model prediction system for obtain de optimal computational resources, at stactic time to mapping parallel task granularity of scientific applications on heterogeneous architectures. This module will be present on future works, because it's required a detailed evaluation process to obtain accurate results.

IV. Computational Resources and Energy Metrics

A. Cluster for Nvidia Tesla GPUs

The computational resources used is GUANE-1[1] cluster, that consists in 16 ProLiant SL390s computing nodes, each node with 2 CPU and 8 GPU for a total of 128 GPU at GUANE. The architecture is detailed in Table I.

B. Server for Intel Xeon Phi

The computational resources used to run the mpHPL is composed by one server with Intel Xeon Phi coprocessor such as shown in Table II.

[1]GUANE-1 (GpU AdvaNced Environment for scientific computing) is a heterogeneous cluster in the High Performance and Scientific Computing Center (SC3-UIS), located at technological camp of Industrial University of Santander at Piedecuesta, a municipality of the metropolitan area of Bucaramanga in Santander, Colombia. See http://www.sc3.uis.edu.co/

Setting GUANE	**A**	**B**	**C**
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Microarchitecture	Sandy Bridge	Sandy Bridge	Sandy Bridge
Processor by node (#)	2	2	2
Clock frequency (GHz)	2.40	2.40	2.670
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
Microarchitecture	Fermi	Fermi	Fermi
GPUs by node (#)	8	8	8
CUDA core (#)	448	448	448
Memory DDR4 (GB)	104	104	104
SAS disk (GB)	200	200	200
Gigabit Ethernet (Gbps)	10	10	10
InfiniBand (IB)	1	1	1

TABLE I: Settings GUANE Cluster Nodes.

Setting PhiServer	**A**
Node type	Server
Number of nodes	1
Processor Intel	Xeon
Processor Model	E5-2630 v3
Microarchitecture	Sandy Bridge
Processor by node (#)	2
Clock frequency (GHz)	2.4 GHz
Core/Processor (#)	8
Thread/Core (#)	2 with HT
Device Intel	Xeon Phi
Device Model	3120A
Microarchitecture	Knights Corner
Device by node (#)	1
Cores per Device	57
Threads per Core	4 with MT
Max Frecuency	1.1Ghz
DeviceMemory	6 GB GDDR5
Memory DDR4 (GB)	64

TABLE II: Settings PhiServer Machine.

C. Energy Metrics Used by EEA Scheme

Instantaneous power over time estimates the overall energy consumption of a system. Where the amount of energy used to achieve the solution, is called *energy-to-solution* in GreenHPC [12]. Energy to solution integrates instantaneous power over time as equation 1 ilustrates.

$$Energy = \int Power(t)\ dt \tag{1}$$

This work proposes the equation 2 to calculate the *energy-to-solution* used in the execution of applications on GUANE Cluster Nodes. The $E_{Solution}$ used the discrete time equivalent of the integral using the power and time samples, for each sample i there is an instantaneous power consumption per node $Power_{Node}(i)$ and a time interval Δt. The total execution time is split in the time used by a group of CPU t_{CPU} and a group of GPU t_{GPU} to proces a percentage of workload the same application. Where N is a maximun execution time of

a application finishes on a group of CPU or a group of GPU.

$$E_{Solution} = \sum_{i=1}^{N} Power_{Node}(i) * max\ (\Delta t_{CPU}(i), \Delta t_{GPU}(i)) \quad (2)$$

The instantaneous power consumption is the throughput of energy delivered on a specific instant [12]. This work estimates the instantaneous power consumption per node $Power_{Node}$ as the sum of each measure of power consumption by component (CPU, GPU and RAM), as represented at equation 3. Where the P^j_{CPU} is the sum of total power consumption by each CPU homogeneous component and nc is the total of CPU. P^j_{GPU} is the sum of total power consumption by each GPU homogeneous component and ng is the total of GPU. P^j_{RAM} is the sum of total power consumption by each main memory and and nm is the total of memories.

$$Power_{Node}(i) = \sum_{j=1}^{nc} P^j_{CPU}(i) + \sum_{j=1}^{ng} P^j_{GPU}(i) + \sum_{j=1}^{nm} P^j_{RAM}(i) \quad (3)$$

For example, to calculate the *energy-to-solution* used to execute a subprogram BLAS3 a matrix multiplication $GEMM$: $C = \alpha * C + \beta * A * B$ using only one CPU (Xeon E5645 2.40 GHz) of 'A' settings GUANE cluster node. The total execution time of GPU t_{GPU} is zero, because they will not be used in this example. The total power per node $Power_{Node}$ must be calculated wiht all components regardless if the application is executed on one CPU or eight GPU and the total execution time at CPU should be calculated at equation 4. The total execution time in a CPU assume overlapping between at the time to move the matrix from memory to cache $time_{comm}$ and the time to perform computation the subprogram $time_{comp}$.

$$t_{CPU} = max\ (time_{comm}, time_{comp}) \quad (4)$$

Therefore if the matrix A and B have a size of $n = 700$ the data size is 3.92 MBytes per matrix. The algorithm complexity to perform computation is $2n^3 FLOP$ and $3n^2$ in memory reference to move the three matrix. The time to move this matrix in chache is $0.367ms$ with $32GB/s$ of bandwith, the time to perform computation is the $11.9ms$ with $52.6GFLOP/s$ of peak and as result the total execution time without uses overlap is $12.267ms$.

The average of power consumption per CPU is represents by $80watts$ when the processor is operating at Base Frequency, the average of power consumption per GPU is represets by $87.21watts$ when the acelerators is operating at Base Frecuency and the average of power consumption per RAM is $70watts$. All those states is called Baseline. In this way, the energy to processes a subprogram is $11.53Kilojoules$ with $940watts$ the total power consumption per node, compared with $0.98Kilojoules$ the energy to processes a subprogram using $80watts$ of power consumption per CPU. This work analysis the power capping technique to manage the power consumption usage by each GPU, enable only the component that is used to perform computation and lowering frequencies to component that is not used.

V. HPL Benchmark as Case of Study

The Highly-Parallel LINPACK (HPL) benchmark solves dense systems $N * N$ of linear equations by partial Gaussian elimination with partial pivoting. The HPL benchmark has a workload of a general dense matrix problem $Ax = b$, using a dense random matrix A. The problem size varies to find the best floating-point execution rate, *Rmax* Maximal LINPACK performance achieved. In computing the execution rate, the number of operations should be $2n^3/3 + 2n^2$ independent of the actual method used [13]. Two types of HPL code are selected to exploit the architectures described on the section IV. The modified variants are described in the following subsections.

A. HPL-2.0 optimized for NVIDIA Tesla GPU

The HPL-2.0 uses in this study case was developed by Petitet [14] is setting for GPU Tesla 20-series and was designed to accelerate the Linpack benchmark on heterogeneous clusters, where CPU and GPU are used in synergy with minor modifications to the original source code. This implementation uses the Intel MKL and CUBLAS libraries to intercepts the calls to DGEMM and DTRSM and executes them simultaneously on both GPU and CPU cores. The HPL-2.0 implementation is described by Fatica [15].

B. HPL-2.1 optimized for Intel Xeon Phi

The HPL-2.1 uses in this study case is part of Intel parallel studio and uses binary accelerates execution by offloading computations to Intel Xeon Phi coprocessors if they are available on the system. This implementation uses the dynamic scheduling technique to highly optimized panel factorization and advanced offload DGEMM. The HPL-2.1 implementation is described by Heinecke [16].

VI. EEA Parameters Used by enerGyPU and enerGyPhi

The enerGyPU and enerGyPhi are a type of batch monitor that is formed by two levels: one level called data capture in runtime composed by three events for data centralization, recording data and start the application. Another separate level called data visualization is used to analyse the results of each experiment in terms of energy efficiency. A deep description of enerGyPU monitor structure and utilization is present by Garcia in [17]

The parameters of the EEA scheme is used by enerGyPU and enerGyPhi monitors to data capture in runtime while is executed in parallel with the HPL code variants on GUANE cluster nodes and Phi-Server. The global parameters are used to setting the workload of a general dense matrix problem size divided in blocks size to generate the total task for processing the $Task_{Number}$ and get the $Task_{size}$ on each experiment. The architectural parameters are the combination of computational resource and configurations used to processing the HPL code variants. The control factors are

monitoring to analizes the implications in power consumption and performance when using different combinations of computational resources to solve the HPL problem. The dependent parameters could be used to created a model that predict the time and power consumption that determine the energy efficiency by specific problem size, described in table III.

Global Workload Parameters	
Ms	Matriz size
Bs	Block size
Architectural Parameters for Nvidia GPU	
GPU_{Usage}	Number of GPU used to HPL
$Cores_{GPU}$	Number of cores per GPU to HPL
MPI_{GPU}	Number MPI process same of GPU used
Architectural Parameters for Intel Xeon Phi	
$Cores_{MIC}$	Number of cores used on MIC
$Threads_{MIC}$	Number of threads per core on MIC
Control Factors for Nvidia Tesla GPU	
SM_{Clock}	GPU SM clock frequency
Mem_{Clock}	GPU memory clock frequency
Control Factors for Intel Xeon Phi	
$Thread_{Affinity}$	Threads execution policy on MIC cores
$Thread_{Granularity}$	Hyperthreading used per core
Dependent Parameters	
$Task_{Size}$	Bytes allocated per task
$Task_{Number}$	Total task for processing
$GPUMem_{Usage}$	GPU memory usage
$MICMem_{Usage}$	MIC memory usage
$Time$	Total time execution of application
$Power_{GPU}$	Total power consumption by all GPU
$Power_{MIC}$	Total power consumption by MIC

TABLE III: EEA parameters used by enerGyPU and enerGyPhi

VII. Experimental Designs and Power Consumption and Performance Analysis

The experimental procedures executed a set test of HPL code variants using different workload and architectural parameters on GUANE cluster nodes and Phi-Server. These results collection of Linpack test allows created a data base with AEE parameters to characterize the power levels. The experimental procedures was chosen following the fractional factorial desing principle propoused by Raj Jain [10] and will described in the next subsections.

A. Experimental Designs to HPL-2.0 on GUANE

The experimental consisting in: 12 different workload parameters using three matriz size $\{49152, 61440, 73728\}$ with four block size $\{768, 1024, 1536, 2048\}$ to divide these matrices. 8 different combinations of architectural parameters such as number of GPU, number of cores for processing, number of cores for administration of GPU and number of MPI processes. 2 control factors (SM and memory clock frequency) that represent tree power levels for GPU worker, GPU idle and GPU baseline or base state.

B. Experimental Designs to HPL-2.1 on Phi-Server

The experimental consisting: 12 experiments with different workload parameters using four matriz size $\{20480, 30702, 40960, 56320\}$ with four block size $\{512, 768, 1024\}$ to divide these matrices. 6 different combinations of architectural parameters such Number of cores used on MIC and Number of threads per core on MIC. 2 control factors as threads execution policy on MIC cores and hyperthreading used per core.

C. Results to HPL-2.0 on GUANE

The arquitectural combination {6 GPU, 2 cores per GPU and 6 MPI process} is the best energy efficient computational resource used in these experiments to solve a matrix size of 49142 and divide with block size of 768, that generates 48 task with $384Mbytes$ of task granularity as shows in the Figure2. This experiment executed the HPL-2.0 in $113.96sec$ with an averange $539.87Watts$ of power cosumption to obtained $694.7GFLOPS$ and also spend $61.52KJoules$ as energy to solution.

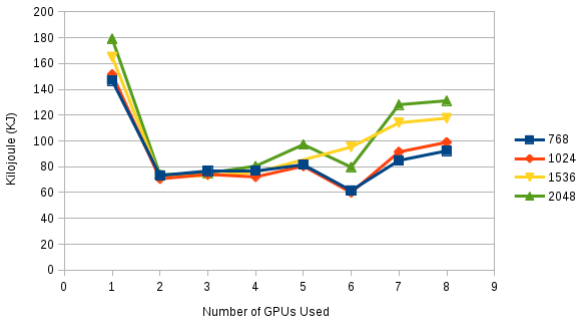


Fig. 2: Ms: 49142 for Resolve the DGEMM on GUANE

The arquitectural combination {4 GPU, 3 cores per GPU and 4 MPI process} is the best energy efficient computational resource used in these experiments to solve a matrix size of 61440 and divide with block size of 768, that generates 80 task with $360Mbytes$ of task granularity as shows in the Figure3. This experiment executed the HPL-2.0 in $222.47sec$ with an averange $435.06Watts$ of power cosumption to obtained $695GFLOPS$ and also spend $96.7KJoules$ as energy to solution.

The arquitectural combination {4 GPU, 3 cores per GPU and 4 MPI process} is the best energy efficient computational resource used in these experiments to solve a matrix size of 73728 and divide with block size of 1024, that generates

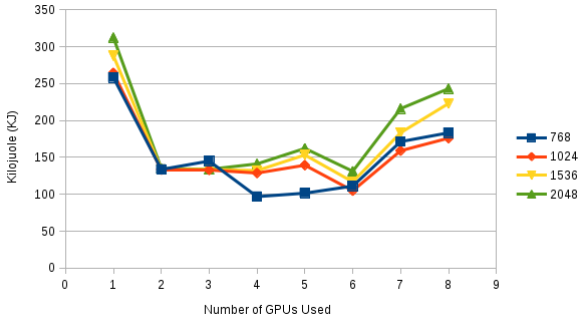


Fig. 3: Ms: 61440 for Resolve the DGEMM on GUANE

72 task with $576Mbytes$ of task granularity as shows in the Figure4. This experiment executed the HPL-2.0 in $327.9sec$ with an averange $435.43Watts$ of power cosumption to obtained $814.8GFLOPS$ and also spend $142.77KJoules$ as energy to solution.

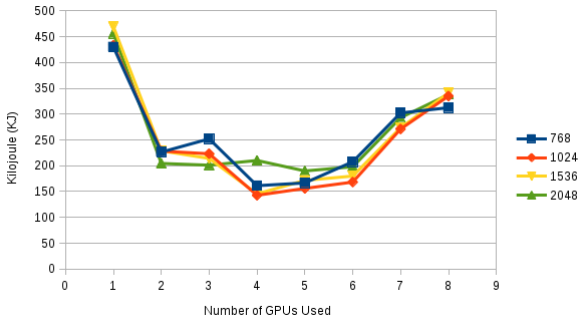


Fig. 4: Ms: 73728 for Resolve the DGEMM on GUANE

D. Power levels on GUANE

The most representative enerGyPU results to solve different matrix problem HPL-2.0 on GUANE are showing at Table IV. This Table is classified in: the best energy to solution (ES), the worst power consumption (PC) and the worst execution time (ET) for each matrix problem. The last two classifications represent the worst energy to solution divided on two behaviors: the first is given for high power consumption determined by the use of all GPU and the second is given for long execution time determined by the use of minimal computational resources.

As a result of using enerGyPU for monitoring performance and power consumption of executed the set test of HPL-2.0 using different workload and architectural parameters on GUANE. Allows obtained the power GPU levels by each determined number of GPU used to process the workload parameters as shows in the Figure 5. The figure represents the total power comsumption for all GPU when are used the power capping in two states GPU worker and GPU idle.

GPU worker: Represents the number of GPU that developer add at architectural parameters, which uses the maximun level of power capping wiht a constant frequencies of 1145 MHz for SM clock and 1566 Mhzz for memmory clock, with a constant memory usage of 2128 MiB, thas generating an averange of power 1120 Watts for each working GPU.

GPU idle: Represents the number of GPU that are inactive, which uses the minimun level of power capping with a constant frequencies of 212 MhZ for SM clock and 340 MHz for memory clock, wich 0% of memory usage to reduce power comsumption to 29.46 watts.

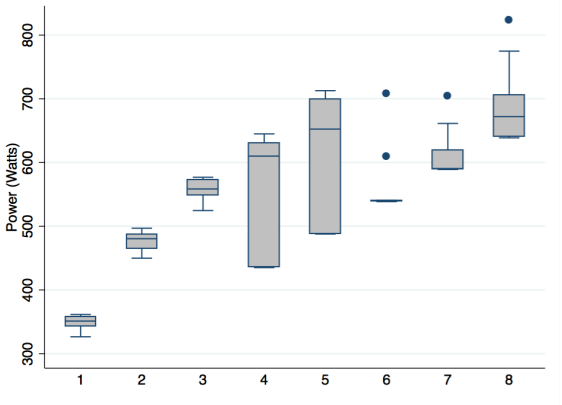


Fig. 5: Power GPU levels

E. Results to HPL-2.1 on Phi-Server

The affinity compact get the best performance and energy efficient computational resource used in these experiments to solve a matrix size of 20480,30720,40960 and divide with block size of 1024the performance is higher, the time to finish is shorter because avoid transfer and synchronization time. as shows in the Figures 6, 7 and 8.

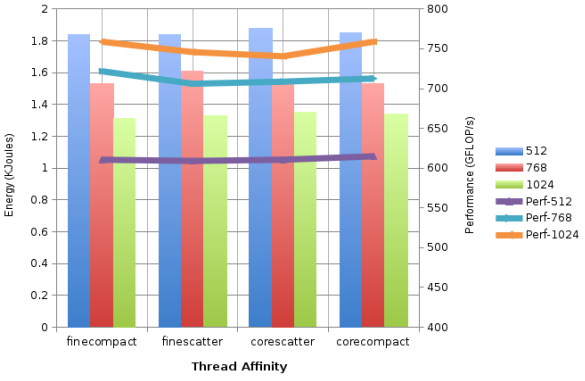


Fig. 6: Ms: 20480 for Resolve the DGEMM on Phi-Server

Experiments to solve HPL2.0	ES	PC	ET	ES	PC	ET	ES	PC	ET
Global Workload Parameters									
Ms	49152			61440			73728		
Bs	768	2048	2048	768	2048	2048	1024	1536	1536
Dependent Parameters									
$Task_{Size}$ *(MBytes)*	384	768	768	360	960	960	576	864	864
$Task_{Number}$	48	24	24	80	30	30	72	48	48
Architectural Parameters for Nvidia GPU									
GPU_{Usage}	6	8	1	4	8	1	4	8	1
$Cores_{GPU}$	2	1	12	3	1	12	3	1	12
MPI_{GPU}	6	8	1	4	8	1	4	8	1
enerGyPU Results									
$enerGyPU_{Time}$ *(sec)*	120	211	537	228	369	915	333	493	1350
$enerGyPU_{L}atency$ *(sec)*	6	6	5	6	6	6	6	6	6
$HPL2.0_{Time}$ *(sec)*	113.96	205.23	531.31	222.47	363.28	909.34	327.9	486.76	1344.09
$Rmax$ *(GFLOPS)*	694.7	131.10	149	695	425.6	170	814.8	548.9	198.8
$Power_{GPU}$ *(Watts)*	539.87	638	337.16	435.06	669.28	337.16	435.43	700.63	337.16
PPW *(MFLOPS/W)*	1286.79	603.93	441.92	1597.44	635.90	494.85	1871.22	783.43	569.37
$E_{Solution}$ ***(KJ)***	**61.52**	**131.10**	**179.13**	**96.7**	**243.13**	**312.38**	**142.77**	**341.04**	**469.25**

TABLE IV: Representative results of enerGyPU to analyze the best energetic to solution (ES), the worst power consumption (PC) and the worst execution time (ET) for each matrix problem.

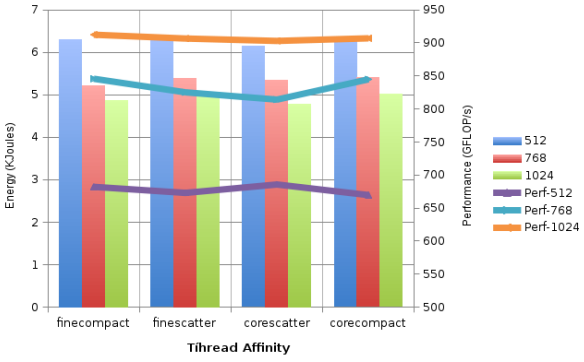


Fig. 7: Ms: 30720 for Resolve the DGEMM on Phi-Server

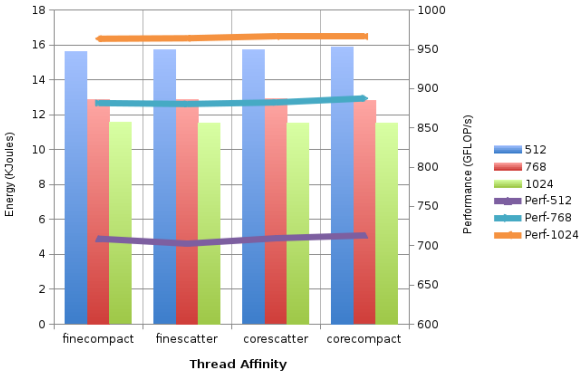


Fig. 8: Ms: 40960 for Resolve the DGEMM on Phi-Server

On high matrices (56320) we observe a benefits of use Hyper-threading (fine) for best performance and energy consumption based on a minor execution time. as shows in the Figures 9.

Thread affinity allows software thread (OpenMP Thread for this test) to execute within in the scope of specific processing resources. In this experiment we using 2 types of thread affinity (compact,scatter) with Hyper-threading enable and disable(fine,core).

When the compact affinity is used each new thread is as close as possible to the thread context, to gain in aspect how cache L2 access on NUMA configuration, the effort of schedule the closer available thread is compensate by the data dependence aspect. For a independent data access the scatter distributes the threads as evenly as possible across the entire system avoid time on schedule process.

The Intel Xeon Phi has UMA architecture (Intel Xeon Phi Coprocessor High-Performance Programming [18] and can use 4 Threads of execution SMT, accepting HT activation.

The Shared Memory (L3) on MIC is 6GB to be used by 57 cores (3120A family), the cost of go to L2 cache is near of 30 clock cycles against 250 clock cycles of use the shared memory (L3), then the thread affinity has tremendous impact on high computations process(Empirical study of Intel Xeon Phi).

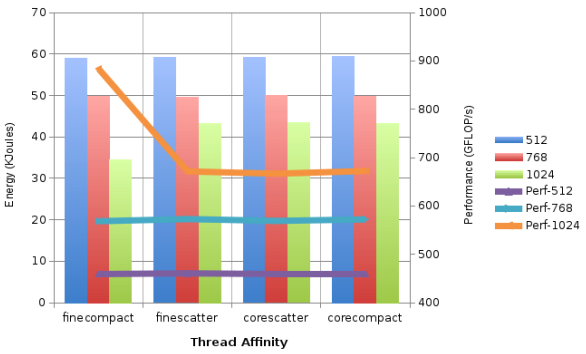


Fig. 9: Ms: 56320 for Resolve the DGEMM on Phi-Server

VIII. Conclusions

We constructed enerGyPU and enerGyPhi to facilitate the collection and visualization of data to analyze many experiments under different combinations of EEA parameters and observe the granularity of the control factors that determine energy efficiency in clusters with multiGPU and server with coprocesadores IntelXeon Phi. The method used in this paper is focused only to analyze previously compiled applications, where researchers do not need to orchestrate the code to execute enerGyPU and enerGyPhi, ensuring the integrity of the results. enerGyPU and enerGyPhi are a software level monitor, therefore can be complemented with other energy monitors that are designed to be plugged-in directly into the power supply to make holistic measures on heterogeneous arquitectures.

Based on the experiment procedures and results presented, the energy-aware EEA scheme is a good alternative to analyze the energy efficiency of applications that runing on arquitectures heterogeneous. The current approach to energy-aware EEA scheme under task-level programming will be use at design on next generation of heterogeneous architecture for large systems, to characterize each specific workload using the best combination of computational resources to optimize the time and power consumption according to task granularity.

IX. Project Repository

To use enerGyPU and enerGyPhi monitors and validate the results, you can download the source code and the data experiments used in this research on project repository url *http://forge.sc3.uis.edu.co/redmine/projects/eea-uis*. To obtain more information about general project you can visit the web site *www.sc3.uis.edu.co*.

X. Acknowledgment

We express gracefull to SuperComputer Center and Scientific Calculation SC3 of Universidad Industrial de Santander for gave access to GUANE GPU cluster and High Performance Computing Center CECAD of Universidad Distrital Francisco Jose de Caldas for offer access to server with Intel Xeon Phi accelerator. The project is developed in collaboration with the High Performance and Scientific Computing Center, SC3UIS of University Industrial of Santander and the Parallel and Distributed Processing Group, GPPD of Federal University of Rio Grande do Sul.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. Williams, and K. Yelick. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead. Technical report, 2008.

[2] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In André Seznec, Uri C. Weiser, and Ronny Ronen, editors, *ISCA*, pages 280–289. ACM, 2010.

[3] Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Commun. ACM*, 56(2):93–102, 2013.

[4] Run Rules, Green500. Energy Efficient High Performance Computing Power Measurement Methodology. Version: 1.2RC2, 2014.

[5] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *ICCD*, pages 349–356. IEEE Computer Society, 2013.

[6] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *ICPP*, pages 48–57. IEEE Computer Society, 2012.

[7] Hongpeng Huo, Chongchong Sheng, Xinming Hu, Baifeng Wu. An Energy Efficient Task Scheduling Scheme for Heterogeneous GPU-Enhanced Clusters. In ICSAI, pages 623-6277. IEEE Computer Society, 2012.

[8] Keliang Zhang and Baifeng Wu. Task Scheduling Greedy Heuristics for GPU Heterogeneous Cluster Involving the Weights of the Processor. In *IPDPS Workshops*, pages 1817–1827. IEEE, 2013.

[9] Alina Sîrbu and Özalp Babaoglu. Power Consumption Modeling and Prediction in a Hybrid CPU-GPU-MIC Supercomputer (preliminary version). *CoRR*, abs/1601.05961, 2016.

[10] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[11] Andrei Radulescu and Arjan J. C. van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. In *Heterogeneous Computing Workshop*, pages 229–238, 2000.

[12] Gustavo Rostirolla, Rodrigo da Rosa Righi, Vinicius Facco Rodrigues, Pedro Velho, and Edson Luiz Padoin. GreenHPC: a novel framework to measure energy consumption on HPC applications. In *SustainIT*, pages 1–8. IEEE, 2015.

[13] Jack Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[14] Antoine P. Petitet. High Performance Computing Linpack Benchmark, HPL - 1.0a. University of Tennessee, Knoxville. Innovative Computing Laboratories. (C) 2000 - 2004.

[15] Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. In David R. Kaeli and Miriam Leeser, editors, *GPGPU*, volume 383 of *ACM International Conference Proceeding Series*, pages 46–51. ACM, 2009.

[16] Alexander Heinecke, Karthikeyan Vaidyanathan, Mikhail Smelyanskiy, Alexander Kobotov, Roman Dubtsov, Greg Henry, Aniruddha G. Shet, George Chrysos, and Pradeep Dubey. Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel Xeon Phi Coprocessor. In *IPDPS*, pages 126–137. IEEE Computer Society, 2013.

[17] John A. G. Henao, Victor M. Abaunza, Philippe O. A. Navaux, Carlos J. B. Hernandez. eGPU for Monitoring Performance and Power Consumption on Multi-GPUs. University Industrial of Santander. In WSPPD, pages 5-8. IEEE Computer Society, 2015.

[18] James Jeffers and James Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

Efficiently Energetic Acceleration EEA-Aware For Scientific Applications of Large-Scale On Heterogeneous Architectures

John A. Garcia. H[1,2,4], Esteban Hernandez B[1,3], Carlos E. Montenegro[3], Philippe O. A. Navaux[2], Carlos J. Barrios H[1]
[1]High Performance and Scientific Computing Center - SC3, Universidad Industrial de Santander - UIS
[2]Parallel and Distributed Processing Group - GPPD, Universidade Federal do Rio Grande do Sul - UFRGS
[3]Interoperability of Academic Networks Group - GIIRA, Universidad Distrital Francisco Jose de Caldas – UD
[4]Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis - I3S, Université Nice Sophia Antipolis - UNS
[1,2,4]*john.garcia1@correo.uis.edu.co, {*[1,3]*ejhernandezb,* [3]*cemontenegrom}@udistrital.edu.co,* [1]*cbarrios@uis.edu.co,* [2]*navaux@inf.ufrgs.br*

***Abstract*— Heterogeneous parallel programming has two main problems on large computation systems: the first one is the increase of power consumption on supercomputers in proportion to the amount of computational resources used to obtain high performance, the second one is the underuse these resources by scientific applications with improper distribution of tasks. Select the optimal computational resources and make a good mapping of task granularity is the fundamental challenge for build the next generation of Exascale Systems. This research proposes an integrated energy-aware scheme called efficiently energetic acceleration (EEA) for large-scale scientific applications running on heterogeneous architectures. The EEA scheme uses statistical techniques to get GPU power levels to create a GPU power cost function and obtains the computational resource set that maximizes energy efficiency for a provided workload. The programmer or load balancing framework can use the computational resources obtained to schedule the map parallel task granularity in static time.**

Keywords—; Energy Efficiency, Power Capping; Energy Aware; Exascale computing

I. INTRODUCTION (*HEADING 1*)

The evaluation of performance and power consumption is a main step in design of applications for large computation systems, such as supercomputers and clusters with nodes that have manycores and multi-GPUs. Researchers must design several experiments for workload characterization to observing the architectural implications of different parameters combinations such as problem size, number of cores per GPU or accelerator, number MPI ranks and observe the resulting clock frequency, memory usage, bandwidth and power consumption, which are factors that determine the performance and energy efficiency of their workload implementation. A key observation from the study of DARPA (Technology Challenges in Achieving Exascale Systems)[1] is that may be easier to solve the power problem associated with base computation than it will be to reduce the problem of transporting data from one site to another on the same chip, between closely coupled chips in a common package or between different racks on opposite sides of a large machine room, or storing data in the aggregate memory hierarchy. Therefore this research designed an integrated scheme called Energy Efficient Acceleration (AEE) as shows in the Figure 1, in which can be used by load balancing frameworks (eg. StarPU, OmpSs) for mapping tasks on the computational resources; then it uses a monitor called enerGyPU and enerGyPhi for capturing metrics of performance and power consumption in order to characterize the application and computing architecture; finally the prediction system uses the scheme AEE to predict the combination of computational resources that maximize energy efficiency of applications running on heterogeneous architectures CPU-GPU type.

II. EEA-AWARE STRUCTURE

A. General Structure

This poster proposes an integrated energy-aware scheme called Efficiently Energetic Acceleration (EEA) for scientific applications on large-scale on heterogeneous architectures like a Figure 1. . The EEA structure has a workflow of three steps as shown in Figure 2.

B. Data capture in runtime

In the first step, the data is captured in runtime and executed the enerGyPU or enerGPhi monitor tool in parallel with the scientific application using different combinations of computational resources, applying the power capping technique for nodes with multi-GPU.

C. Data visualization and statistics characterization

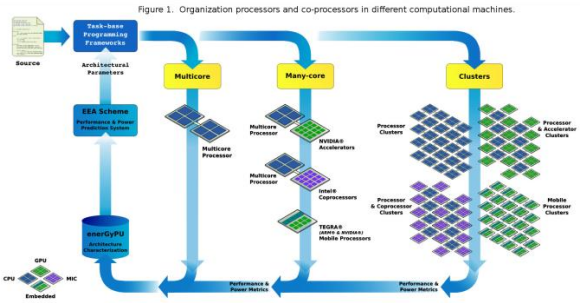
In the second step, the data visualization and statistics characterization used a separate level of enerGyPU and enerGPhi monitor tool for analysis the key factors by results of each experiment in terms of energy efficiency and estimated the power levels. A deep description of monitor structure and utilization is present by García John et al. in [2], [3].

D. Model prediction system

Finally using the data collected by monitors cost functions are build and model prediction system running for obtaining the optimal computational resources in a static time to mapping parallel task granularity of scientific applications on heterogeneous architectures.

III. Experimental procedures and results

At the first level the global parameters have chosen the workload and computational architecture according to the combination of resources that will be used. The experimental procedures were executed with a set of tests of HPL code variants using different workload and architectural parameters on Cluster nodes GUANE. The experimental procedures were chosen following the fractional factorial design principle proposed by Raj Jain [4]. The source code may be download from our repository locate on stable version on http://forge.sc3.uis.edu.co/redmine/projects/eea-uis and testing version on https://github.com/jagh/enerGyPU



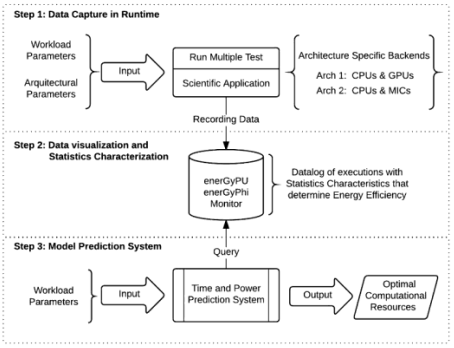
IV. Conclusions

Each computational problem need a characterization to get the best resource distribution on each specific workload, this process required a huge time to tuning the best resource distribution, special when need improve the energy use. This research found that use that using huge number of task with relative few data size offer an efficient energetic acceleration scheme to solve linear equations on heterogeneous architecture with GPU capable cards. enerGyPu and enerGyPhi were develop to help the characterization process (capture, analyzing and visualization) across of several experiments with different resource distributions and so determine the control factor that offer the best energetic efficiency without change the applications with instrumentation process.

Additionally, we develop the EEA scheme to choose the computational resource needed by a specific workload on runtime and then use a workload manager that could implements this metrics (like StartPU or OmpSs) and apply power capping reducing the energy consumption and improve the performance of solution.

With this work we show that need improve the current metrics using for the traditional workload manager and jobs schedules to incorporate new energy-aware metrics like energy-to solutions o performance per watt, because it's the key factor to building exascale architectures.

This Research demonstrate that using a huge number of task with a small data size, it allow obtain an energetic efficient acceleration when processed linear equation systems on nodes with heterogeneous architecture CPU-GPU type



Acknowledgment

We express graceful to Supercomputer Center and Scientific Calculation SC3 of Universidad Industrial de Santander for gave access to GUANE GPU cluster; to the High Performance Computing Center CECAD of Universidad Distrital Francisco Jose de Caldas for offer access to server with Intel Xeon Phi accelerator; and the Mining Data (MinD) research group is part of the Software and Knowledge Engineering Department of the I3S Laboratory on Computer Science, Signals and Systems of Sophia-Antipolis. I3S is a mixed research unit of the University of Nice Sophia-Antipolis for gave access to Octopus GPU node cluster.

References

[1] K. Bergman et al. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead. Technical report, 2008.

[2] John A. García H. et al. enerGyPU and enerGyPhi Monitor for Power Consumption and Performance Evaluation on Nvidia Tesla GPU and Intel Xeon Phi. CCGrid 2016 - IEEE/ACM.

[3] John A. García H. et al. eGPU for Monitoring Performance and Power Consumption on Multi-GPUs. XIII Workshop de Processamento Paralelo e Distribuído, WSPPD 2015 - IEEE.

[4] Raj jain. The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley, 1991.

enerGyPU for Monitoring Performance and Power Consumption on Multi-GPUs

Super Computación y Cálculo Científico UIS

John A. García. H.[1,2], Philippe O. A. Navaux[2], C.J. Barrios Hernández[1]

[1,2] john.garcia1@correo.uis.edu.co, [2] navaux@inf.ufrgs.br, [1] cbarrios@uis.edu.co

[1.] High Performance and Scientific Computing Center, SC3UIS
Universidad Industrial de Santander - Bucaramanga, Colombia

[2.] Parallel and Distributed Processing Group, GPPD
Universidade Federal do Rio Grande do Sul - Porto Alegre, Brasil

Abstract

enerGyPU is a tool that can be used to analyze multiple tests under different combinations of parameters to observe the key factors that determines the energy efficiency in terms of ***'Energy per Computation'*** *on Clusters with Multi-GPUs. enerGyPU is a monitor to centralize and automate the data captured in runtime while is executed in parallel with the scientific application and displays information via sequence plots, statistical tables, bar graphs and shows results in terms of energy efficiency.*

Background and Motivation

The evaluation of performance and power consumption is a key step in the design of applications for large computation systems, such as supercomputers, clusters with nodes that have manycores and multi-GPUs. Researchers must design several experiments for workload characterization by observing the architectural implications of differents combinations of parameters; such as problem size, number of cores per GPU, number MPI of process and observe the resulting clock frequency, memory usage, bandwidth, and power consumption, which are factors that determine the performance and energy efficiency of their workload implementation. The major problem in performance evaluation consists on experiment design given so many parameters. If the resulting data is not properly collected and the parameters of evaluation performance are not well organized, the design exploration may achieve wrong conclusions [1].

enerGyPU is developed to get accurate measurements of many experiments that are need for the project called ***"Energy Efficient Acceleration of Large-Scale Scientific Applications on Heterogeneous Architectures"***, where the goal is determine the distribution of tasks of large-scale scientific applications on a CPU-GPU heterogeneous architecture, for an efficient computational acceleration and low power consumption. This approach allows to obtain the optimal number of computational resources such as GPU, cores per GPU and number of Process MPI at a static time, in which a developer or framework of load balance can configure the number of computational resources to maximize the energy efficiency. This project is developed in collaboration by the High Performance and Scientific Computing Center, SC3UIS and Distributed Processing Group, GPPD.

enerGyPU Monitor Structure

enerGyPU is a type of batch monitor that is formed by two levels. One level is called ***enerGyPU to data capture in runtime***, which is executed in parallel with the scientific application, and is composed by three events. A separate level called ***enerGyPU to data visualization online***, that can be used later for analyses of results of each experiment in terms of energy efficiency, and is composed by four events.

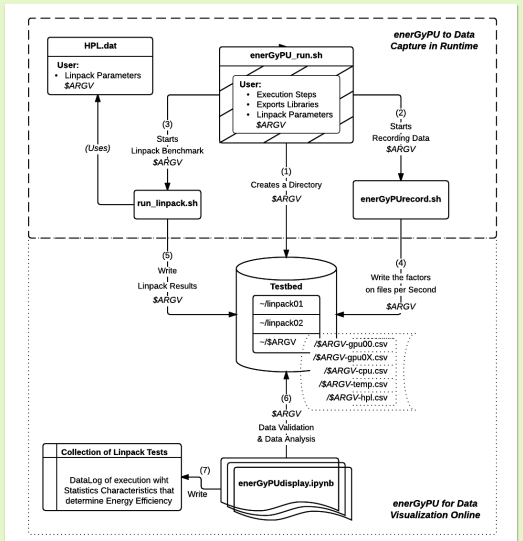


Figure 1. enerGyPU Monitor Structre

The first level is used to run a script called ***enerGyPUrun.sh*** where execution steps have been declared, and the libraries needed to run scientific, We used the Linpack Benchmark as study case. The user writes the same Linpack parameters defined in the file ***HPL.dat*** to execute the experiment. Those parameters are used as a unique key called ARGV to share the argument in all the following events. ***enerGyPUrecord.sh*** is executed in background for data extraction and writing: one file per GPU parameter (Streaming Multiprocessor Clock Frequency-SM, Memory Clock Frequency, Memory Usage, Power Consumption). New data registered every second.

The second level is used at post-processing for data visualization. It displays information via sequence plots, statistical tables, histograms and shows results in terms of energy efficiency. The event (6) displays information of tests through the IPython Notebook Viewer in a program called ***enerGyPUdisplay.ipynb*** that contains predefined code routines written in Python, where the researchers have to type the argument ARGV to identify the unique test of the Linpack benchmark.

Experimental Procedures and Results

This section presents the use of enerGyPU to observe the key factors that determine the energy efficiency in one test with Linpack benchmark. We used HPL.2.0 version configured for Tesla GPUs [3]. The Linpack parameters used are shown in Table 1. The computational resources used for this experiment are part of one node of the 'A' settings of cluster GUANE that consists in ProLiant SL390s-G7 computing nodes such as shown in Table 2.

Matrix size	49152
Block size	1024
GPU Used	4
Cores per GPU	3
Process MPI	4

Table 1. The Linpack parameters Used in experimental procedure.

Setting GUANE	A	B	C
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Processor by node (#)	2	2	2
Clock frequency (GHz)	2.40	2.40	2.670
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
GPUs by node (#)	8	8	8
Memory DDR (GB)	104	104	104
SAS disk (GB)	200	200	200

Table 2. Settings GUANE Cluster Nodes.

This experiment procedure shows the 1 GPU of 4 GPU that are inactive, and other 2 GPUs of 4 GPUS actually working have constant frequencies of 1147MHz for SM clock and 1566MHz for memory clock, with a constant memory usage of 2128MiB, thus generating an average power of 120 watts between working GPUs. The results obtained for performance was 691.20 GFLOPS, for one node of Cluster GUANE achieved the energy ef- ficiency of 1097.68 MFLOPS/W.

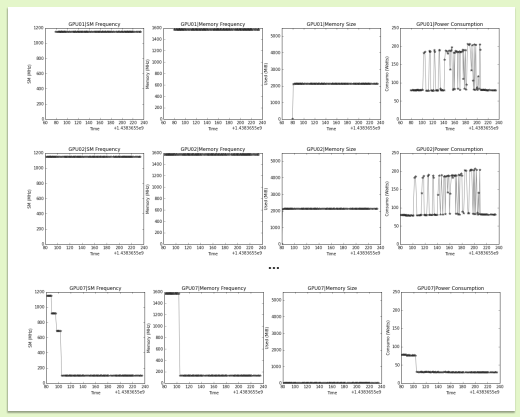


Figure 2. enerGyPU-Sequenceplot of Power Consumption by each GPU

References

[1] Raj Jain. The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley, 1991.

[2] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Electrical Engineering and Computer Science Department, University of Tennessee, University of Manchester, 2014.

[3] Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. In David R. Kaeli and Miriam Leeser, editors, GPGPU, volume 383 of ACM International Conference Proceeding Series. ACM, 2009.

More Information:
www.sc3.uis.edu.co/
www.inf.ufrgs.br/gppd

Project Repository:
https://jagarcia@forge.sc3.uis.edu.co/eea-uis/energypu.git

Energetically Efficient Acceleration EEA-Aware For Scientific Applications of Large-Scale On Heterogeneous Architectures

John A. García. H.[1,2,4], Esteban Hernandez B.[1,3], Philippe O. A. Navaux[2], Carlos. J. Barrios H.[1]

[1,2,4] henao@i3s.unice.fr, [1,3]ejhernandezb@udistrital.edu.co, [2]navaux@inf.ufrgs.br, [1]cbarrios@uis.edu.co

1. High Performance and Scientific Computing Center, SC3 Universidad Industrial de Santander, UIS - Bucaramanga, Colombia
2. Parallel and Distributed Processing Group, GPPD Universidade Federal do Rio Grande do Sul, UFRGS - Porto Alegre, Brasil
3. Interoperability of Academic Networks Group - GIIRA, Universidad Distrital Francisco Jose de Caldas, UD - Bogota, Colombia
4. Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis - I3S, Université Côte d'Azur, UNA - CNRS - Nice, France

Challenge

Heterogeneous parallel programming has two main problems on large computation systems:

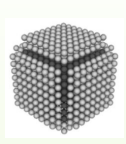
- The increase of power consumption on supercomputers in proportion to the amount of computational resources used to obtain high performance.
- The underuse these resources by scientific applications with improper distribution of tasks.

"Select the optimal computational resources and make a good mapping of task granularity is the fundamental challenge for build the next generation of Exascale Systems"

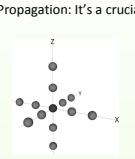
Analysis of Energy Efficiency and Portability of large Scale Scientific Application:

Case Study: Ondes3D©

Simulation of Seismic Wave Propagation: It's a crucial tool for analyzing geophysical strong movements.



3D Seismic Model with blocks discretization



Spatial Data Dependencies Stencil

Unknowns:

Velocity and Stress
On each point of Space and Time.

$$\frac{\partial v_i}{\partial t} = \frac{\partial \sigma_{ij}}{\partial j} + F_i \quad (3.6)$$

Seismic Wave Equation for a solid medium.

$$\frac{\partial \sigma_{ij}}{\partial t} = \lambda \delta_{ij} \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right) + \mu \left(\frac{\partial v_i}{\partial j} + \frac{\partial v_j}{\partial i}\right) \quad (3.7)$$

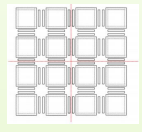
Constitutive relation in the case of an isotropic medium.

Ref. John A. García H. et al. Energetically Efficient Acceleration EEA-Aware. Degree work to obtain the title of Master of Science in Systems Engineering and Informatics at UIS 2016.
Ref. Víctor Martínez et al. Towards Seismic Wave Modeling on Heterogeneous Many-Core Architectures Using Task-Based Runtime System. IEEE PAD 2015.

Planning workload experimental on Ondes3D©

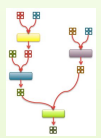
StarPU© task programming library for hybrid architectures.

7.2 Millions Grid (300×300×80) - 625 MB Block Size 150x150

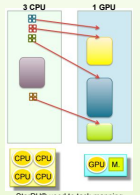


2D seismic model discretized in stencil blocks of primary grid.

36 Task processing (20 Velocity; 16, Stress) 144 Task of communication



Task dependency: Directed Acyclic Graph (DAG)



StarPU© used to task mapping over CPU and GPU

Ref. John A. García H. et al. Energetically Efficient Acceleration EEA-Aware. Degree work to obtain the title of Master of Science in Systems Engineering and Informatics at UIS 2016.
Ref. Víctor Martínez et al. Task-based programming for Low-power Manycore Architectures: Seismic Simulation on Jetson TK1. CARLA 2015.
Ref. Olivier Aumage. Porting BRGM's Ondes3D on top of StarPU. Task Parallelism on Heterogeneous platforms. INRIA – University of Bordeaux. March 2013.

Heterogeneous Architectures

Results of running Ondes3D© with StarPU© on different machines.

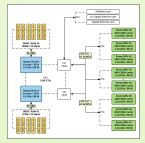
Jetson TK1

Mean Power Consumption on Jetson TK1(15 Watts):
10 Watts by GPU use.
5 Watts by CPU use.

Desktop: Salmon

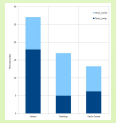
Mean Power Consumption of Salomon Desktop (135.6 Watts):
Limite Superior de la GPU: 191.35 Watts (Processing)
Nivel medio de GPU: 80 Watts (Idle)

Node: GUANE

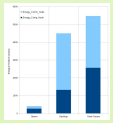


Mean Power Consumption on GUANE Node(640 Watts)
Upper limit of GPU Node: 1600 Watts (Processing)
Mean level of GPU Node: 640 Watts (Idle)

Time to Solution



Energy Consumed on Execution



Jetson(27.06 seconds):
17.97 seconds in processing tasks.
9.09 seconds in transfer tasks.

Desktop(16.94 seconds):
4.03 seconds in processing tasks.
11.36 seconds in transfer tasks.

Node (13.23 seconds):
6.20 seconds in processing tasks.
7.03 seconds in transfer tasks.

Jetson(405.9 Joules):
270.6 Joules by using GPU.
135.3 Joules by using CPU.
10W (GPU) + 5W (CPU) = 15 Watts

Desktop(4580.5 Julios):
2298.3 Joules by using GPU.
2282.2 Joules by using CPU.
135.6W (GPU) + 130W (CPU) = 265.6 Watts

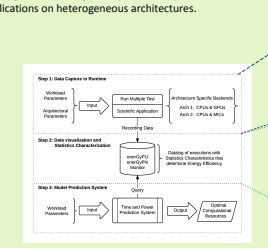
Node (5470.4 Julios):
2692.1 Joules by using CPU y GPU.
2778.3 Joules by component in an idle state.
203.4W (N_run) + 210W (N_Idle) = 413 Watts

Which is the task distribution to be mapping on heterogeneous architectures CPU-GPU, that determine an efficient computational acceleration and low power consumption during the time of scientific application execution?

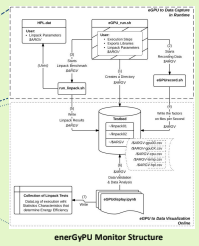
Ref. John A. García H. et al. Energetically Efficient Acceleration EEA-Aware. Degree work to obtain the title of Master of Science in Systems Engineering and Informatics at UIS 2016.

EEA-Aware Structure

This research proposes an integrated energy-aware scheme called Efficiently Energetic Acceleration (EEA) for scientific applications of large-scale on heterogeneous architectures. The EEA energy-aware scheme has a workflow of three steps as shown in EEA-Aware Scheme. In the first step, the data is captured in runtime and executed the enerGyPU or enerGPhi monitor tool in parallel with the scientific application using different combinations of computational resources, applying the power capping technique for nodes with multi-GPUs. In the second step, the data visualization and statistics characterization used a separate level of enerGyPU and enerGPhi monitor tool for analysis the key factors by results of each experiment in terms of energy efficiency and estimated the power levels. A deep description of monitor structure and utilization is present by García John et al. in [1], [2] and [3]. Finally using the data collected by monitors cost functions are build and model prediction system running for obtaining the optimal computational resources in a static time to mapping parallel task granularity of scientific applications on heterogeneous architectures.



EEA-Aware Scheme



enerGyPU Monitor Structure

Step 1: Experimental Procedures and Results

At the first level the global parameters have chosen the workload and computational architecture according to the combination of resources that will be used. The experimental procedures were executed with a set of tests of HPL code variants using different workload and architectural parameters on Cluster nodes GUANE. The experimental procedures was chosen following the fractional factorial design principle proposed by Raj Jain [4].

The computational resources used: 1 GUANE Node on 'A' settings.

Configuraciónes	A	B	C
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Microarchitecture	Sandy Bridge	Sandy Bridge	Sandy Bridge
Processor by node (#)	2	2	2
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
Microarchitecture	Fermi	Fermi	Fermi
GPUs by node (#)	8	8	8
CUDA core (#)	448	448	448
Memory DDR4 (GB)	104	104	104
SAS disk (GB)	200	200	200
Gigabit Ethernet (Gbps)	10	10	10
InfiniBand (IB)	1	1	1

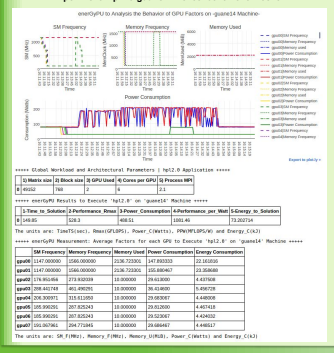
The Linpack used: HPL.2.0 version configured for Tesla GPUs.

Ref. Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. ACM, 2009.

Experiments to solve HPL2.0	ES	PC	ET	ES	PC	ET	ES	PC	ET
Global Workload Parameters									
Mx	49152			61440			73728		
By	768	2048	2048	768	2048	2048	1024	1536	1536
Dependent Parameters									
$Task_{Size}$ (MBytes)	384	768	768	360	960	960	576	864	864
$Task_{Number}$	48	24	24	80	30	30	72	48	48
Architectural Parameters for Nvidia GPU									
GPU_{Num}	6	8	1	4	8	1	4	8	1
$Cores_{GPU}$	2	1	12	3	1	12	3	1	12
MPI_{GPU}	6	8	1	4	8	1	4	8	1
enerGyPU: Results									
$enerGyPU_{Time}$ (ms)	120	211	537	228	369	915	333	493	1350
$enerGyPU_{efficiency}$ (ms)	6	6	5	6	6	6	8	6	6
$HPL2.0_{Time}$ (sec)	113.96	265.25	531.31	222.47	363.29	909.34	327.9	490.76	1344.09
$Rmax$ (GFLOPS)	694.7	333.18	149	695	425.6	170	814.6	538.9	198.8
$Power_{GPU}$ (Watts)	539.97	639	337.19	435.99	669.29	387.39	435.41	700.03	337.19
PPW (MFLOPS/W)	1296.79	603.58	441.92	1597.44	635.99	484.85	1871.22	765.41	593.87
$E_{Solution}$ (KJ)	61.52	131.39	179.13	96.7	243.13	312.38	142.77	341.04	469.25

Representative results of enerGyPU to analyze **the best Energetic to Solution (ES)**, **the worst Power Consumption (PC)** and **the worst Execution Time (ET)** for each matrix problem.

Experiment: hpl2.0-guane14-20150730316-49152768262



***** Global Workload and Architectural Parameters | hpl2.0 Application *****

Matrix size	Block size	Cores per GPU	Process MPI
49152	768	2	2.1

***** enerGyPU Results to Execute 'hpl2.0' on 'guane14' Machine *****

Time_to_Solution	Performance_Rmax	Power_Consumption	Performance_per_Watt	Energy_to_Solution
120.85	639.3	468.51	1.365	73.282734

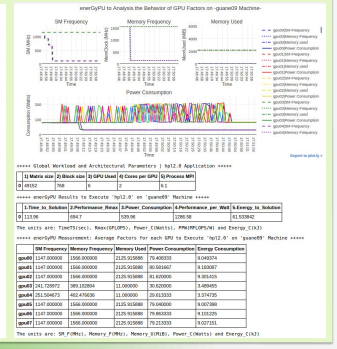
The units are: Time(Sec), Rmax(GFLOPS), Power_C(Watts), PPW(MFLOPS/W) and Energy_C(KJ)

***** enerGyPU Measurement: Average Factors for each GPU to Execute 'hpl2.0' on 'guane14' Machine *****

	SM Frequency	Memory Frequency	Memory Used	Power Consumption	Energy Consumption
gpu00	1147.000000	1566.000000	2136.723301	147.893333	22.165356
gpu01	1147.000000	1566.000000	2136.723301	155.880467	23.356680
gpu02	176.951466	270.832039	10.000000	29.613800	4.437438
gpu03	398.441748	491.498291	10.000000	36.414690	5.456728
gpu04	206.309971	315.611093	10.000000	29.683697	4.448008
gpu05	195.950291	297.825243	10.000000	29.012600	4.467418
gpu06	185.906291	287.825243	10.000000	29.523067	4.424332
gpu07	191.007961	294.771845	10.000000	29.686667	4.448517

The units are: SM_F(Mhz), Memory_F(Mhz), Memory_U(MiB), Power_C(Watts) and Energy_C(KJ)

Experiment: hpl2.0-guane09-20150731117-49152768626



***** Global Workload and Architectural Parameters | hpl2.0 Application *****

Matrix size	Block size	Cores per GPU	Process MPI
49152	768	6	2

***** enerGyPU Results to Execute 'hpl2.0' on 'guane09' Machine *****

Time_to_Solution	Performance_Rmax	Power_Consumption	Performance_per_Watt	Energy_to_Solution
113.96	639.3	509.96	1.365	81.523842

The units are: Time(Sec), Rmax(GFLOPS), Power_C(Watts), PPW(MFLOPS/W) and Energy_C(KJ)

***** enerGyPU Measurement: Average Factors for each GPU to Execute 'hpl2.0' on 'guane09' Machine *****

	SM Frequency	Memory Frequency	Memory Used	Power Consumption	Energy Consumption
gpu00	1147.000000	1566.000000	2125.925888	79.408333	9.540324
gpu01	1147.000000	1566.000000	2125.925888	80.581667	9.160087
gpu02	1147.000000	1566.000000	2125.925888	81.626000	9.301435
gpu03	241.728972	383.132804	11.000000	30.630000	3.489403
gpu04	251.504673	402.475836	11.000000	29.613333	3.374735
gpu05	1147.000000	1566.000000	2125.925888	78.682500	8.972190
gpu06	1147.000000	1566.000000	2125.925888	79.960333	9.101235
gpu07	1147.000000	1566.000000	2125.925888	79.213333	9.027151

The units are: SM_F(Mhz), Memory_F(Mhz), Memory_U(MiB), Power_C(Watts) and Energy_C(KJ)

Step 2: Architectural Characterization

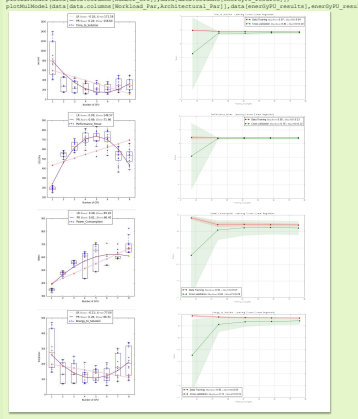
The second level uses the enerGyPU monitor in the post-processing for data visualization and statistical characterization of each architecture. In which the Figure below shows that the accuracy using the key factors is much more accurate than just using the number of GPUs. In addition, this monitor displays information via sequence data, statistics, histograms and tables showing results in terms of energy efficiency, for each experiment.

Minimization model with Task and Architectural Parameters

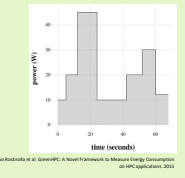
`GPUPowerRuntime(GPU_worker[GPU_Factors],GPU_idle[GPU_Factors])`

Accuracy of Linear and Polynomial Models Among Learning Curves Multivariable Model

```
plotLinModel(data[data.columns[Number_GPU]],data[data.columns[energyGPU_results]])
plotMultModel(data[data.columns[Workload_Par,Architectural_Par]],data[energyGPU_results],energyGPU_results)
```



Discretization Power Consumption per Unit Time:



Ref. Gustavo Rostirolla et al. GreenHPC: A Novel Framework to Measure Energy Consumption on HPC applications. 2015

Equations to Calculate the Energy Consumption:

$$Energy = \int Power(t)\, dt \quad (3.1)$$

$$Energy = \sum_{i=1}^{N} Power(i) * \Delta t(i) \quad (3.2)$$

$$E_{Solution} = \sum_{i=1}^{N} Power_{Node}(i) * max\left(\Delta t_{CPU}(i), \Delta t_{GPU}(i)\right) \quad (3.3)$$

$$Power_{Node}(i) = \sum_{j=1}^{80} P^{j}_{CPU}(i) + \sum_{j=1}^{78} P^{j}_{GPU}(i) + \sum_{j=1}^{80} P^{j}_{RAM}(i) \quad (3.4)$$

$$\Delta t_{CPU}(N) = max\left(time_{comm}, time_{comp}\right) \quad (3.5)$$

Ref. John A. García H. et al. Energetically Efficient Acceleration EEA-Aware. Degree work to obtain the title of Master of Science in Systems Engineering and Informatics at UIS 2016.
Ref. Jack Dongarra. Algorithmic and Software Challenges: For Numerical Libraries at Exascale, 2013.

Step 3: EEA Prediction System

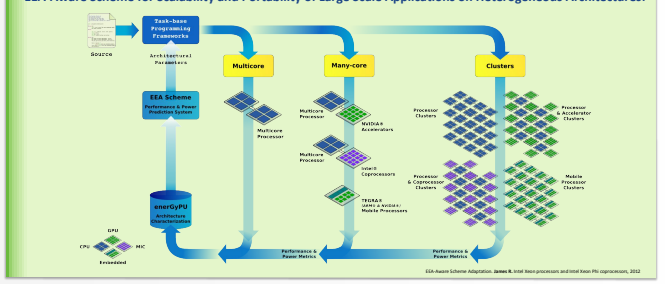
The third level uses the projected multivariable regression model results to see metrics such as time, performance, power consumption, power consumption and performance per watt, which are used to execute the HPL for each combination of computational resources and can calculate the best combination of computational resources, as shown in Figure.

***Efficiently Energetic Acceleration* | EEA Prediction System**

```
In [25]: eeaPredictionSystem(49152)
***** EEA Prediction System of Computational Resources for Matrix 49152 to Execute HPL-2.0 on GUANE *****
Out[25]:
```

	Best Prediction	Time_to_Solution	Performance_Rmax	Power_Consumption	Energy_to_Solution	Performance_per_Watt
block_size	768.000000	1024.000000	768.000000	2048.000000	1024.000000	768.000000
Number_task	64.000000	48.000000	64.000000	24.000000	48.000000	64.000000
Task_size	288.000000	384.000000	288.000000	768.000000	384.000000	288.000000
GPU	3.000000	6.000000	3.000000	1.000000	6.000000	3.000000
Cores	4.000000	2.000000	2.000000	12.000000	4.000000	4.000000
Time_to_Solution	89.730042	13.185518	56.792579	751.901756	50.122982	93.730342
Performance_Rmax	662.160353	640.140940	674.374812	118.356452	627.940492	662.160353
Power_Consumption	646.216050	609.774292	607.899581	356.528464	548.092760	646.216050
Energy_to_Solution	55.831045	41.340581	50.205452	241.895280	31.870272	56.802045
Performance_per_Watt	1.218755	1.080187	1.151112	0.406359	1.147830	1.218755

EEA-Aware Scheme for Scalability and Portability of Large Scale Applications on Heterogeneous Architectures:



EEA-Aware Scheme Adaptation. James R. Intel Xeon processors and Intel Xeon Phi coprocessors, 2012

More Information:
www.sc3.uis.edu.co/
www.i3s.unice.fr/

Project Repository:
Stable:
http://forge.sc3.uis.edu.co/redmine/projects/eea-uis
Testing:
https://github.com/japh/enerGyPU

References

[1] **John A. García H. et al.** Energetically Efficient Acceleration EEA-Aware. Degree work to obtain the title of Master of Science in Systems Engineering and Informatics at UIS 2016.

[2] **John A. García H. et al.** enerGyPU and enerGyPhi Monitor for Power Consumption and Performance Evaluation on Nvidia Tesla GPU and Intel Xeon Phi. CCGrid 2016 - IEEE/ACM.

[3] **John A. García H. et al.** eGPU for Monitoring Performance and Power Consumption on Multi-GPUs. XIII Workshop de Processamento Paralelo e Distribuído, WSPPD 2015 - IEEE.

[4] **Raj jain.** The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley, 1991.

[5] **K. Bergman et al.** Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead. Technical report, 2008.