Arquitecturas Heterogeneas Y Sus Retos.

Computacion Heterogenea.

EDWAR ANDRES VILLARREAL FORERO
2122970
Bucaramanga/Santander
Tiendanet1000@gmail.com

Resumen

La computación heterogénea es alusiva a sistemas en donde se tienen mas de un tipo de procesador para desarrollar las determinadas funciones. Esta clase de sistemas esta desarrollada para generar mas rendimiento en base a el manejo de diferentes tipos de procesadores. Normalmente se incorporan métodos de procesado particulares para cada una de las tareas a efectuar.

En computación la palabra heterogeneidad hace referencia a distintos tipos de arquitecturas de conjuntos de instrucciones comúnmente denominado (ISA).

Abstract.

Heterogeneous computing systems is alluding to have systems where more than one type of processor to develop specific functions. This kind of system is developed to generate more performance based on handling different types of processors. Usually they incorporate specific methods for each of the tasks to perform processing.

In computing the word heterogeneity refers to different types of set architectures commonly referred instructions (ISA) .

I. COMPUTACION HETEROGENEA

La heterogeneidad se ha visto gradualmente aumentada a causa de la minimización de los componentes , estos a su vez causaron un aumento en la tecnología basada en chips, esto lo conocemos como system –on-chip. Un ejemplo de esto lo podemos notar en los nuevos procesadores los cuales tienen en su estructura una lógica encargada de interactuar con otros dispositivos de el mismo sistema como : SATA , Ethernet , USB etc….

En computación la palabra heterogeneidad hace referencia a distintos tipos de arquitecturas de conjuntos de instrucciones comúnmente denominadas (ISA). esta heterogeneidad no solo hace referencia a tener un procesador principal el cual posee un conjunto de instrucciones y el resto poseen otra diferentes sino que se aplica tambien a sistemas en los cuales la velocidad esta basada principalmente en las diferentes microarquitecturas internas de la misma ISA.

Los recientes descubrimientos nos muestran que un chip multiprocesador de ISA-heterogénea que posee múltiples ISAs en su arquitectura, aumentara el rendimiento de la mejor arquitectura heterogénea con ISAs del mismo tipo en un 21%. Además de un ahorro de consumo de energía del 23%.

.

II. RETOS DE LA COMPUTACION HETEROGENEA

Los Sistemas de computación heterogénea presentan diferentes retos sin solución aun hoy en día.

La creación de maquinas para el procesamiento simultaneo de múltiples datos aumenta todos los problemas derivados de los sistemas homogéneos los cuales usan procesamiento en paralelo. No obstante , el nivel de heterogeneidad en el sistema puede introducir una determinada situación comúnmente llamada no-uniformidad en sistemas de desarrollo, de esto podemos obtener prácticas de programación y además también mejoraremos las capacidades del sistema global.

Las diferentes áreas en las cuales se puede introducir nuestra definición de heterogeneidad estarán brevemente contempladas en los siguientes sub índices , lo que se intenta mostrar al lector es la increíble capacidad de hacer cambios notorios en la velocidad de procesamiento basándonos en los distintas formas de ensamblar por asi decirlo los componentes básicos de un procesador , a esta definición se le puede añadir otra opción optima pero basada en otra arquitectura de ordenamiento.

Los diferentes retos que debe enfrentar la heterogeneidad son:

1) ISA:

una ISA es un conjunto de instrucciones , esta detalla tdas las instrucciones que puede entender una unidad central de procesamiento o tambien puede ser el conjunto de instrucciones implementados por un diseño particular de un ordenador.

Este termino hace referencia a los aspectos principales del procesador visibles para un programador. existen tres tipos :

- CISC(complex instruction set computer)
- RISC(reduced instruction set computer)
- SISC(simple instruction set computing)

Esto abre la posibilidad de que procesadores con diferentes diseños y arquitecturas como es el caso de Intel Pentium y AMD Athlon compartir un conjunto de instrucciones , por ejemplo: los anteriores mencionados implementan modelos casi idénticos del conjunto de instrucciones x86 pero como dijimos anteriormente tienen diferentes diseños.

Los distintos elementos de computacion peden tener en su estructura arquitecturas de conjuntos de instrucciones diferentes, esto puede ocasionar una incompatibilidad en la parte binaria.

2) ABI:

También es conocida como interfaz binaria de aplicaciones y es la encargada de ser la interfaz entre normalmente un sistema operativo o una librería y los lleva a lenguaje maquina , es decir binario.

Podemos ver implementadas las ABIs en:

- Disposición y tamaño de los tipos de datos
- Alineamiento de los tipos de datos

Otras ABIs tienen detalles mas estandarizados como la integración de nombres de funciones en c++. .

Los distintos elementos internos de computo o procesamiento pueden interpretar la memoria de diferentes maneras , esto incluye manejo de memoria etc.. , esto depende tanto de la arquitectura como del compilador utilizado.

3) API:.

Abreviadamente es conocida como API (application programming interface) en español interfaz de programación de aplicaciones. Esta es un conjunto de funciones, métodos, procedimientos o subrutinas que nos proporciona una determinada biblioteca para ser utilizada a nuestro antojo.

En otras palabras, un API sirve para actuar como enlazador entre los diferentes componentes del software, es decir un intercomunicador. Algunos ejemplos de estas interfaces de programación son :

- Microsoft Win32 API
- CORBA(Common Object Request Bróker Architecture)
- Microsft WMI

los servicios y librerias que tiene internamente el sistema opertivo puede no estar disponible de forma uniforme y funcional para todos los elementos de computo, esto es debido a la variedad o diversidad en los componentes del mismo; esto es a lo que llamamos heterogeneidad o arquitecturas heterogeneas.

4) Interfaz de memoria y jerarquia:

Los sistemas dependiendo de su fabricante pueden tener diferentes estructuras internas de cache , en los cuales podemos ver algunas mejoras entre uno u otro sistema . tambien se puede notar la variedad en los protocolos de coherencia de la cache.

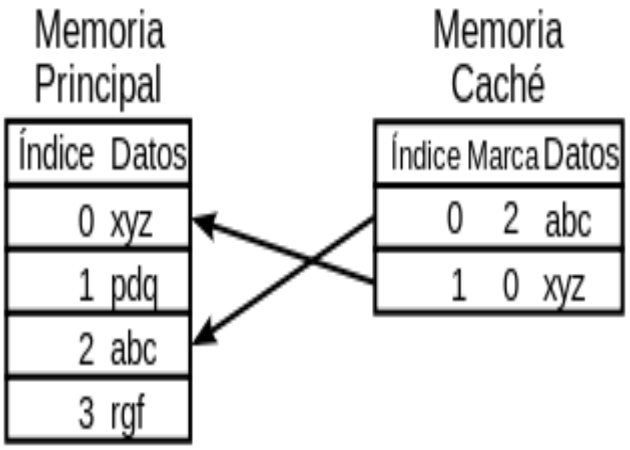
Cache:

la cache es la memoria a la cual se puede acceder rápidamente en una computadora, esta guarda de manera temporal los datos que están siendo utilizados por la computadora.

La memoria cache es también denominada búfer , y funciona de manera semejante a la memoria principal pero esta es de un tamaño menor y accesible de manera rápida para el equipo.

Una arquitectura para mejorar el procesado de datos es la de ubicar la cache entre la unidad central de procesamiento(CPU) y la memoria de acceso aleatorio (RAM) , esta estructura se usa para acelerar el intercambio de datos.

A continuación veremos una forma esquematizada de ver la estructura interna de una cache y su proceso básico interno:



El acceso a esta cache puede ser de manera uniforme o no , a esto ultimo se conoce como (NUMA).

NUMA: (Non Uniform Memory Access)

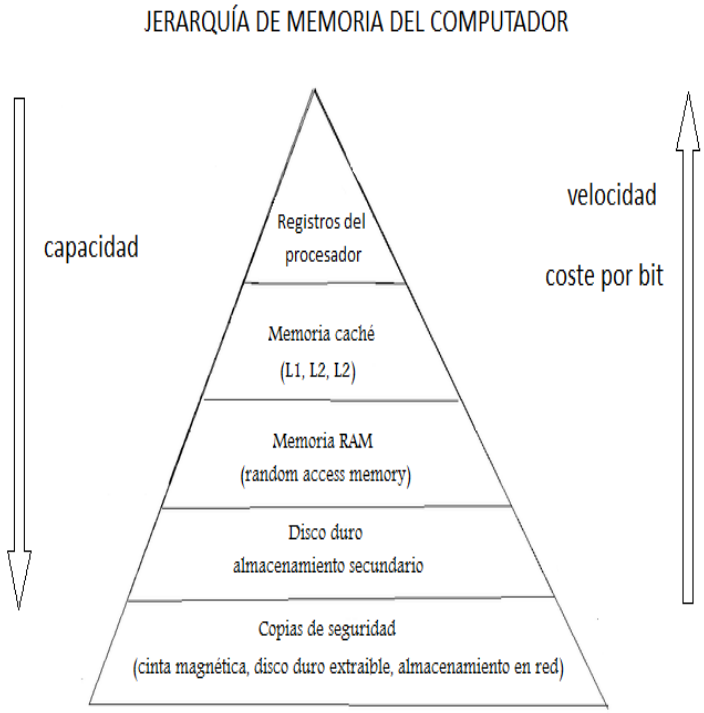
En español acceso a memoria no uniforme. es un diseño de memoria donde se accede a esta en posiciones relativas de otro proceso o se comparte la misma memoria entre procesos.

Podemos ver cuantiosas diferencias en la habilidad de cada una para leer datos escogidos de manera arbitraria . algunos procesadores realizan accesos de un byte , una palabra , o a mayor cantidad.

JERARQUIA:

La jerarquía de memoria es la organización en forma de pirámide de la memoria, esta se basa en subniveles de organización. Este sistema es utilizado por las maquinas.

Para esquematizar mejor esta arquitectura notemos el siguiente grafico:



Las siguientes afirmaciones se cumplen para el diseño de memoria:

- A mayor capacidad menor costo por bit
- A mayor capacidad menor velocidad
- A menor tiempo de acceso mayor coste

Lo que se busca en esta parte de el sistema es poseer la cantidad suficiente de memoria y a su vez tener una velocidad suficiente para cumplir con los estándares requeridos y mas importante , que sea a un precio accesible.

5) IMPLEMENTACION DE BAJO NIVEL DE LAS CARACTERISTICAS DEL LENGUAJE.

Hay otras caracteristicas tales como hilos de ejecucion o funciones previamente definidas ,normalmente se implementa utilizando punteros a funcion. Este mecanismo representa un nuevo reto ya que requiere de traducciones adicionales cuando se esta utlizando en un entorno de heterogeneidad.

Estas traducciones adicionales se pueden ver en forma abstracta , esta ultima es la forma en la cual un programador determina su estructura interna.

6) INTERCONEXION:

Consiste en una técnica que responde a la necesidad de hacer posible la interacción entre distintas infraestructuras o redes con otras tecnologías y diseños diferentes. La finalidad de estas aplicaciones es poder percibirlo como si se tratara de una sola red interconectada.

Los elementos con los cuales trabajamos habitualmente pueden acomodarse o bueno tener distintas formas de conectarse entre sí, a esto se conoce como interconexión. Además de esto, también se tienen interfaces graficas de memoria (BUS).

Los retos basados en interfaces son mucho mas amplios que los elementos con los que realizamos nuestra conexión entre dispositivos.
Los dispositivos de acceso de memoria directa son parte de esto:

Acceso directo a memoria:
También conocido por sus siglas (DMA) del inglés Direct Memory Access, permite a distintos componentes de un ordenador acceder a la memoria del sistema para escribir o leer independientemente de la unidad de procesamiento o bueno la CPU. Muchos hardware se basan en DMA donde incluyen controladores de unidades de discos, también se incluyen tarjetas de sonido y tarjetas gráficas.
Esto se ha convertido en una característica esencial de todos los ordenadores modernos ya que esto permite que dispositivos de diferentes velocidades puedan comunicarse sin someter a la CPU a una constante carga de procesos creando interrupciones en sus tareas principales.

Problemas de coherencia entre la DMA y la memoria cache:

La DMA puede ocasionar problemas de coherencia en la cache.
Imaginemos una CPU que viene con una memoria cache y también una memoria externa a la cual se puede acceder por los dispositivos que utilizan DMA, Cuando la CPU accede a W lugar en memoria, el valor actual se almacena en la cache. Si después se realizar operaciones en W, se actualizara la copia de cache de W, pero la versión de memoria externa de W no será actualizada si la cache no se borra en memoria antes de que otro dispositivo acceda a W; el dispositivo recibirá un valor caducado de esta misma variable.

Estos problemas se generan por falta de un proceso específico. Y gracias a estos "problemas" los cuales se solucionan basando estos mismos como retos ; ya sean retos personales o empresariales , podemos encontrar soluciones prácticas y hasta soluciones ideales u optimas para los problemas tratados.

Algunos problemas tratados los podemos abordar diseñando el sistema de la siguiente forma:

Los sistemas de cache coherente:
Estos implementan un método externo en el hardware del dispositivo y por medio de este se escribe una señal en el controlador de la cache y esta a su vez invalida la cache para escritura DMA.

Sistemas no coherentes:
Este sistema deja esto al software , en donde el sistema operativo debe asegurarse de vaciar las líneas inscritas en cache ante de que una transferencia de salida de DMA sea iniciada y anularla antes de que una parte de la memoria sea afectada por la entrada de DMA que se halla requerido.

El sistema operativo también debe evitar que esa parte de memoria no sea afectada por cualquier subproceso que se ejecute en ese instante. Este ultimo es un poco menos eficiente ya que crea cierta redundancia en las operaciones de la DMA ya que la mayoría de hardware requiere un bucle para invalidar cada línea de cache de forma individual.

Podemos ver arquitecturas hibridas basadas en este reto o problema. Donde en la cache secundaria llamémosla J2 es coherente, en cambio la cache J1 es manipulada o administrada por el software. J1 normalmente es la CPU.

Un claro ejemplo de esta situación es la encontrada en un ordenador que posee arquitectura ISA con controlador de DMA el cual está basado en Intel 8237, este es un controlador de DMA multimodo, el cual es una combinación de hardware y software. Estas son arquitecturas hibridas las cuales han estado presentes desde los inicios de la era digital. uno de los grandes pioneros de la innovación es IBM ; en antiguos equipos podíamos notar un controlador DMA capaz de ofrecer cuatro canales DMA, estos realizan la transferencia de ocho bits y solo pueden dirigirse al primer megabyte de RAM.

First in First out (FIFO):
En español seria primero en entrar primero en salir, este concepto es utilizado en estructuras de datos y también en teoría de colas.
Análogamente podríamos abordar este tema con un ejemplo cotidiano. Tenemos un grupo de personas que están esperando en una cierta fila y serán atendidas o procesadas en el modo en que llegaron, es decir que la primera persona que llego será la primera persona en salir.
Si nos basamos más en estructura de datos, FIFO es utilizado para implementar colas. Este método podemos llevarlo al desarrollo computacional por medio de arreglos y vectores o también por medio de asignación de punteros y asignación dinámica de la memoria .

Si implementamos esto por medio de vectores, el número máximo de elementos que podría almacenar nuestro FIFO estaría limitado por el código que se haya generado anteriormente antes de la compilación (habitualmente conocida como cola estática) o durante su ejecución (cola

dinámica). Sea cual sea nuestra elección el número de elementos que podrá almacenar la cola estará previamente establecida antes de la ejecución del programa , no una variable , la cual seria mas practica. Debido a esto el sistema deberá reservar previamente el espacio en memoria necesario para poder guardar todos los datos, no importa el número de elementos usados.

Debido a esto surge un nuevo reto para el avance; en algunas aplicaciones esto genera un problema ya que puede desconocerse el número de elementos que se van a introducir a la cola. La simple solución de reservar mas memoria de la que se supone que se necesitara podría costear un desperdicio de memoria , sin embargo si se usa asignación dinámica en memoria el número máximo no estaría determinado por tiempo de compilación sino por tiempo de ejecución , esto significa que se reserva memoria a medida que esta se vaya necesitando , es decir que se vaya expandiendo la cola. Esta simple arquitectura genera una solución optima para evitar esa perdida de memoria innecesaria.

Los retos son la base de a innovación.

Resumiendo un poco hablare de algunas porciones de un sistema heterogéneo pueden tener coherencia de cache mientras que otros necesitan una actuación explicita de la parte del software para poder mantener la coherencia y la consistencia en el sistema.

7) RENDIMIENTO:

Nos basaremos principalmente en el hecho de que un sistema heterogéneo puede tener CPUs similares en términos de arquitectura, pero tener diferencias particularmente subyacentes en su micro arquitectura que conllevan a varios niveles de rendimiento y consumo de energía.

De los retos que presentan las arquitecturas hibridas en esta parte del tema, uno de los que genera as controversia es el impedimento energético hacia la creación de maquinas de computación cada vez mas potente pues a energía para hacer funcionar una maquina de estas magnitudes es bastante elevada , tan elevada que es casi absurdo gastar la energía de una planta nuclear cercana solo para generar un poco lapso de súper computo; se puede decir que esto será un reto futuro , un reto tanto para las arquitecturas heterogéneas como para la tecnología en general y finalmente llevara a la innovación futura.

El rendimiento es utilizado en múltiples aspectos y lo podemos conseguir de múltiples maneras. hablando de arquitectura podemos acelerar cálculos de estructura electrónica mediante el uso de arquitecturas de procesadores especializados en el computo masivo. Partiremos de LIO , una implementación existente de los algoritmos vistos en la teoría de los funcionales de la densidad por su siglas(DFT) ya que esta hacia uso de placas de video para computo general (GPGPU). Este programa se adaptó para usar todas las opciones ofrecidas por multiprocesadores, GPUs modernas etc…
El cuestiones de código se las mejoras se enfocaron en código computacionalmente mas intenso conocido como el calculo de la energía de intercambio y correlación. En las GPUs se busco cambiar la estructura de paralelizacion en la estructura de computo y aprovechar la mayor cantidad de memoria de las placas para almacenar mas resultados intermedios.
La implementación realizada en las CPUs y Xeon Phi es compartida, aprovechando al máximo las opciones prestadas por el compilador para realizar vectorizacion y paralelismo de manera práctica.

Los resultados de optimización obtenidos en Xeon Phi resultan bastante inferiores en comparación con otras arquitecturas existentes, pero esta fue la base para identificar puntos claves de coprocesador que abrirán las puertas a mejorar la optimización ya existente.

En las GPUs se desarrolló e implemento el uso de múltiples placas de video en una misma plataforma, su ejecución es de manera escalada linealmente en función de la cantidad de los dispositivos usados. Adicionalmente notamos que a implementar esta arquitectura en CPUs y GPUs genera un rendimiento complementario con respecto a las tareas necesarias para el calculo de la energía de intercambio y correlación. Esto nos lleva a concluir que podemos lograr mejoras con una implementación hibrida CPU-GPU utilizando el hardware ya creado.

GPGPU:
Es un nuevo concepto dentro de la informática y esta basado en arquitecturas bastante interesantes , debido a esto quiero hablar un poco sobre este nuevo concepto que trata de estudiar y aprovechar las capacidades de computo de una GPU.
Una GPU es un pequeño procesador diseñado para cómputos basados en gráficos 3d interactivo. Tiene la particularidad de ser un procesador de bajo costo si nos basamos en relación a su potencia de calculo.

Gracias a estos estudios hemos generado bases de datos , simulaciones de fluidos y particularmente algoritmos de clustering.

Debido a las diferencias entre las arquitecturas de una GPU y una CPU no cualquier problema se puede beneficiar de una implementación en una GPU.
Específicamente hablando, el acceso a memoria plantea las mayores dificultades. Las CPU están diseñadas para el acceso aleatorio en memoria.. Esto ayuda a la creación de estructuras de datos complejas, con punteros a posiciones aleatorias en la memoria. En cambio, en una GPU, el acceso a memoria está más restringido. Por ejemplo, en un procesador de vértices (la parte de una GPU creada para transformar vértices en

aplicaciones 3D), se favorece el modelo scatter, en el que el programa lee en una posición predeterminada de memoria, pero escribe en una o varias posiciones aleatorias. En cambio, un procesador de pixeles favorece el modelo gather, pudiendo el programa leer varias posiciones aleatorias, pero escribir en sólo una posición predeterminada.

Anteriormente , el desarrollo de software GPGPU se había hecho muy bien en lenguaje ensamblador , o bueno en algún lenguaje especifico para aplicaciones graficas usando la GPU, algunos ejemplos de esto son GLSL, HLSL , etc….
Pero recientemente han emergido herramientas para facilitar este trabajo, al hacer abstracto muchos de los detalles relacionados con los gráficos y presenta una interfaz de mas alto nivel.

Una de estas herramientas es Sh , una extensión del conocido c++ para algo llamado metaprogramacion con una implementación automática a a GPU.

La opción mas comercial y mas utilizada en la actualidad es la basada en CUDA , una tecnología de NVIDIA. Es una extensión de c que permite la codificación de algoritmos en la GPU.

Compute Unified Device Architecture(CUDA):

Esta arquitectura es de vital importancia en este trabajo ya que es una muestra de el arte de las arquitecturas heterogéneas, CUDA surge naturalmente de la aplicación de el hardware desarrollado para problemas gráficos pero enfocados en el computo científico.

Estas placas de video se empezaron a implementar a partir de 1978 con la invención de Intel del chip ISBX 275.
En 1985 se había incluido un coprocesador grafico que podía ejecutar instrucciones independientes de la CPU, este fue un gran paso a la separación de tareas. En los noventa diversos avances surgieron a partir de la tecnología 2D y para mediados de la década ya se estaban desarrollando tecnologías 3D. en el 2000 se agregaron los shaders a las placas , eran unos programas pequeños que corrían internamente en la GPU, lo particular de esto es que se podían encadenar entre si , uno por pixel en la pantalla , este paralelismo fue el que llevo a as GPUs a poder procesar operaciones graficas en magnitud mucho mas rápido que e CPU.

En el 2006 NVIDIA llega innovando con la arquitectura G80 el cual es la primer GPU que deja de preocuparse por resolver únicamente problemas gráficos para pasar a un motor genérico que cuenta con una serie de instrucciones preprogramadas, esta arquitectura es la que se ha adoptado y mejorado durante el tiempo permitiendo a las GPUs tener procesadores mas simples de baja frecuencia de reloj y con una disipación térmica manejable.

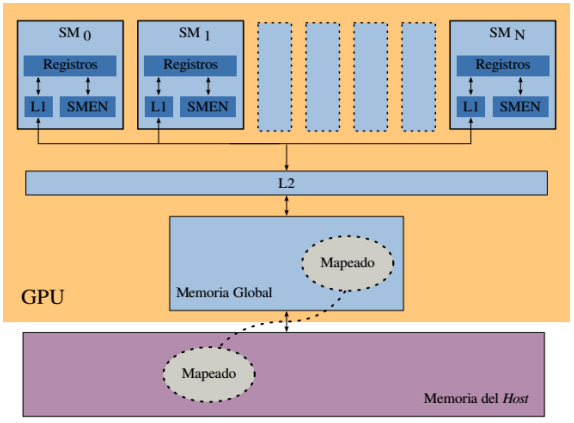
Para poder explotar a arquitectura CUDA los programas deben ser diseñados de manera que el problema se pueda particionar usando el modelo de grilla de bloques de Threads.

Hoy en dia se cuenta con nuevas soluciones referentes a portabilidad que se siguen desarrollando , bibliotecas como OpenMP , Y OpenACC son herramientas que buscan generar códigos que se puedan utilizar de manera eficiente el acelerador de computo que se halla disponible. Estas herramientas permiten desarrollar las operaciones de manera genérica y dejan el trabajo pesado a la parte del compilador para que subdivida el problema o “reto” de manera que el acelerador ya sea la CPU , GPU o MIC necesite.

Otro aspecto que ha sido desarrollado basándose en los diferentes retos que se generan al intentar generar un cambio han sido logrados mediante la organización de procesadores: NVIDIA ha diseñado procesadores GPGPU que han sido reorganizados a lo largo de su existencia múltiples veces pero conservan algunas características en su diseño a través de su transformación.

Ahora nos concentraremos en una ejemplificación mas figurativa de algunos de los mágicos componentes que hacen posible esta arquitectura.

Esquema de la jerarquía de memorias en una GPU:



- Cache l2
- Cache constante
- Cache de textura

La cache l1 esta representada por SM, la cache l2 es común a todos los SM de la GPU, la cache constante es una cache sobre la memoria global dedicada únicamente a lecturas de memoria y la cache de textura es una cache sobre la memoria total y cuenta con la particularidad de que se le puede agregar el concepto de dimensiones para poder modelar datos en mas de una dimensión.

REQUISITOS DE UN PROBLEMA PARA LAS GPGPU:

Un problema basándonos en un procesador GPU debe poseer las siguientes características mínimas para que este posea potencialidad para poder aprovechar al máximo las características de esta arquitectura:

Primero: el problema a tratar necesita una gran parte paralelizable. Esto se refiere a que debe existir una manera practica de partir el problema en subrutinas o subproblemas que puedan realizarse de manera simultánea sin que halla alguna dependencia de resultados entre si.

Segundo: el problema debe basarse en su mayoría de operaciones numéricas.

Tercero: el problema debe ser modelable utilizando arreglos o matrices.

Cuarto: El tiempo de transferencia de datos debe ser inferior al tiempo de computo.

Si intentamos agrupar todas estas ideas , todas estas distintas formas de organizar , de operar o de ejecutar determinadas tareas para mejorar la ejecución de la misma, son notables cambios en la arquitectura de nuestros sistemas y estos a su vez representan una superación a las distintas incógnitas que surgieron durante esta investigación para al final terminar superando esos retos , esas adversidades y lograr algo nuevo, novedoso , innovador.

PLATAFORMAS DE COMPUTACION HETEROGENEA:

Estas plataformas son encontradas en cada ámbito de la informática desde maquinas de alto rendimiento de calculo y servidores dedicados hasta dispositivos embebidos de bajo consumo como las tablets y los teléfonos celulares.
Ahora enunciaremos algunos de ellos y ejemplos particulares de cada una de ellas , algunas serán dejadas como manera de pregunta para el lector:

Computacion reconfigurable:

- Intel (Atom , Altera FPGA)

Tecnologia de propósito general:

- Juegos y dispositivos de entretenimiento
- APUs AMD
- IBM Cell , principalmente usado en playstation 3
- Emotion Engine , encontrado en el playstation2

Redes:

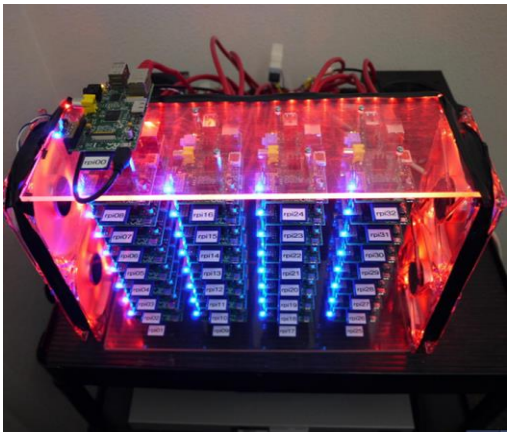
- Procesadores de red Intel IXP
- Procesadores de red Netronome NFP.

Sistemas embebidos:

- Texas Instruments OMAP
- Dispositivos analógicos BACKFIN
- Nvidia Tegra
- Apple A.

Ejemplo de un sistema embebido con multiples salidas y conexiones basadas en una arquitectura especifica.

Ahora nos adentraremos un poco en una función muy interesante de estos sistema , en especial basándonos en la raspberry pi y este tiene que ver con la parte de supercomputación:



Lo que aquí vemos es una practica en ingeniosa forma de usar una configuración ingeniosa basados en la Raspberry pi.

Xeon Phi:

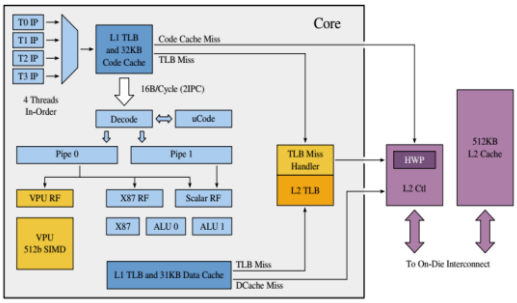
Esta arquitectura es la culminación de un trabajo que se inicio en Intel en el 2004 , prediciendo la necesidad de paralelismo masivo para aplicaciones futuras , esta sale al mercado al finales del 2012 con el único propósito de competir en computo intensivo con Nvidia y su CUDA.

Xeon Phi ha sido implementado en la supercomputadora Tianhe-2 de la universidad de sun Yat-Sen en china , la cual apareció en el top 500 de las computadoras mas rapidas del mundo en distintos años; los 16000 nodos de esta supercomputadora posee en su armazón dos Ivi Bridge Xeon y tres coprocesadores Xeon Phi cada uno , logrando en conjunto una capacidad total de computo teorico de 54,9 PetaFLOPS.

Microarquitectura :

Durante la creación de Xeon Phi se tuvieron en cuenta distintos factores , uno de estos es el consumo energético. Uno de los objetivos fue aumentar el poder de computo por watt de los procesadores Xeon de la época.

La microarquitectura de este procesador esta basada en mas de 50 procesadores simétricos que comparten memoria lo cual hace referencia a su nombre (MIC) MANY INTEGRATED CORE.



Esquema de un procesador del Xeon Phi.

CONCLUSIONES:

En cada uno de los casos anteriores se concluía al finalizar cada uno de los mecanismos utilizados.

Los temas que hemos abordado durante todo el articulo están basados en arquitecturas heterogéneas y lo que finalmente damos a entender es que toda composición, unión, modificación y mejora de alguna función de el sistema o ya sea memoria , cache o interno del procesador hace un aporte notorio en la velocidad de procesamiento de un equipo , esto puede verse reflejado desde diversos puntos de abstracción . todos estos dispositivos creados recientemente surgieron en base a que de una problemática establecida se pueden generar múltiples soluciones , si implementamos esto a nuestro concepto.

También podemos concluir que una parte vital de estas arquitecturas y de estos grandes retos establecidos fue el descubrir que con diferentes formas de estructurar los componentes de el procesador generamos una velocidad considerable , y que día a día estos retos son superados llevando consigo la innovación y la proposición de nuevos retos a intentar resolver. Esta ideología es lo que nos ha impulsado a alcanzar todo en lo cual hace algunos años ni siquiera se sabia que existía y unos veinte años mas ni siquiera sabíamos que era esta definición. Me impresiona como el trabajo en equipo y de la mano de la industria se ha podido hacer mucho as avanzada la tierra en cuestión de días , tal vez semanas.
Los avances tecnologicos no tendrán limite?
Nuestra imaginación no tiene limites.

BIBLIOGRAFIA:

Shan , Amar (2006) , heterogeneus processing : a strategy for augmenting Moore´s law.

AMD Announces proyect skybridge .

Computación heterogenea
Retos de las arquitecturas heterogéneas. Información guía tomada de :
https://es.wikipedia.org/Computacion_heterogenea

Tesis sobre arquitecturas multifuncionales.
Capitulo4: CUDA.
Documentación guía tomada de:
https://www.dc.uba.ar/inv/tesis/licenciatura/2015/