Clusters. Programación en Clusters
Algebra Lineal en Paralelo sobre Clusters

Fernando G. Tinetti
fernando@info.unlp.edu.ar
Curso de Postgrado
Facultad de Informática, UNLP
50 y 115, 1900, La Plata
Argentina

2010

Clusters. Programación en Clusters
Algebra Lineal en Paralelo sobre Clusters

UNLP - 2010

1. Mencionar los tres problemas más importantes que resolvió utilizando computadoras.
2. Identificar el contexto en el cual resolvió cada problema (producción, proyecto de investigación, proyecto de cátedra, interés personal).
3. Indique qué aprendió de cada uno de los problemas.
4. Enumere los pasos que siguió para resolver cada problema.
5. Indique si está en un proyecto de investigación actualmente e indicar cuáles son los objetivos a corto, mediano y largo plazo desde su punto de vista (en caso de estar en un proyecto).

Nombre:

Cargo/Posición:

Universidad/Lugar de Trabajo:

1 Introducción

Contenido

- Programa/Detalles del Curso publicado
  - Objetivos
    - Conceptos de programación distribuida.
    - Análisis de los modelos de comunicación entre procesos y procesadores.
    - Soluciones basadas en bibliotecas de pasaje de mensajes tipo MPI.
    - Clusters homogéneos y heterogéneos.
    - Caracterización del modelo de arquitectura de cluster.
    - Resolución de problemas numéricos y no numéricos sobre clusters.
    - Identificación de las características de rendimiento paralelo en general y de las específicamente relacionadas con el rendimiento paralelo en los clusters.
    - Identificación de las ventajas de utilizar clusters en general para cómputo paralelo.
    - Identificación de las posibles penalizaciones de rendimiento en los clusters.
    - Identificación de las herramientas básicas para análisis de rendimiento sobre clusters.
  - Programa
    - Unidad I. Introducción.
    - Unidad II. Ideas de procesamiento intensivo extraídas del área de aplicaciones de álgebra lineal. Análisis de problemas no numéricos.
    - Unidad III. Arquitecturas de procesamiento: desde las computadoras paralelas específicas hasta los clusters.
    - Unidad IV. Bibliotecas de cómputo numérico/matricial secuenciales y paralelas.
    - Unidad V. Problemas y soluciones para operaciones con altos requerimientos de cómputo. Análisis de problemas y soluciones posibles.
    - Unidad VI. Extensiones a más de un cluster ¿Más problemas que soluciones?
    - Unidad VII. Análisis de posibles trabajos finales.
  - Duración
    - 20 horas de clase presencial.
    - 40 horas de trabajo fuera de clase.
    - 2 horas para exponer en forma individual los proyectos.
  - Modo de Evaluación
    - Proyectos de trabajo individual con 3/6 meses para presentarlos.

- Temas recurrentes
  - Cómputo de Alto Rendimiento (HPC).
  - Algebra lineal.
  - Paralelismo en multiprocesadores y en multicomputadoras.
  - Evaluación de rendimiento secuencial y paralelo.
  - Arquitecturas de procesamiento paralelo.
  - Bibliotecas de software disponibles.
  - La mayor cantidad de práctica posible.
  - Los temas de otros cursos.
- Levantando el nivel de abstracción (si pudiéramos):
  - Soporte de sistemas operativos.
  - Bibliotecas en general.
  - Abstracciones intercluster.
  - Internet computing.
  - Grid-Cloud computing...
  - ...

Organización/Modalidad

	Original	*Actual*
Clases	Lunes	Todos los días de una semana
Práctica	Trabajos indivuduales por escrito Todos los lunes, con conclusiones	Algunas cosas/taller ¿Todos los días?
Aprobación	Proyecto individual	Proyecto individual

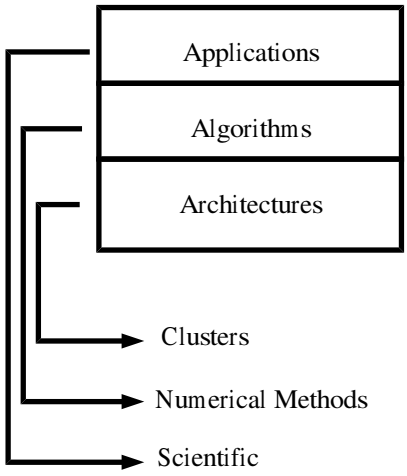
Las prácticas serán tanto de consolidación de temas como de explicación/introducción de temas.

Conocimientos previos (deberían):

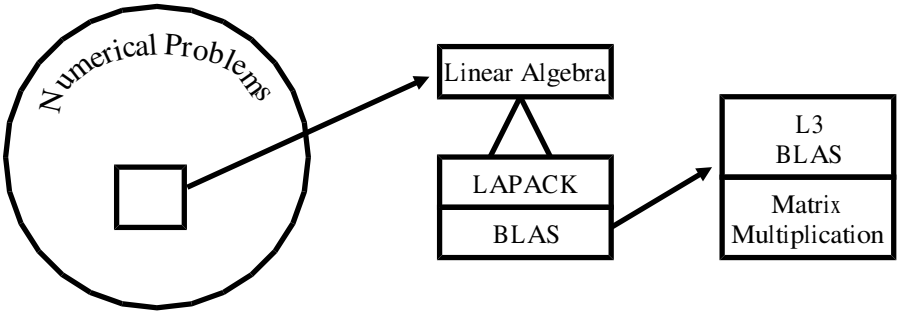
- Conocimientos "mínimos" de álgebra.
- Ideas de Cómputo paralelo.
- Análisis de algoritmos, análisis de rendimiento (arquitecturas).

Siempre hay muchos problemas...

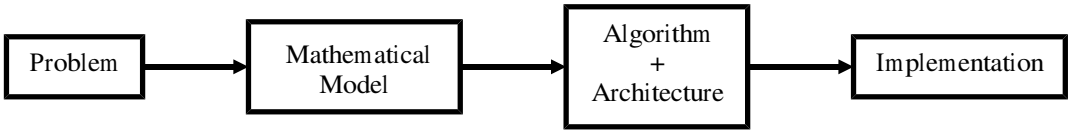
Software en Producción:



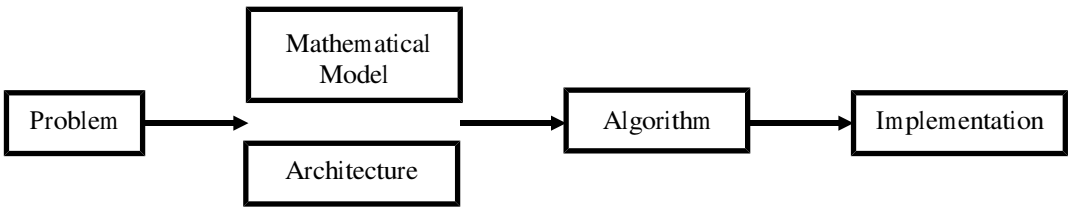
Visión más algorítmica:



Para llegar al código en producción:



Sin embargo, para llegar al código en producción (2):



Muchos algoritmos han sido propuestos y utilizados, con énfasis en:

- Modelos matemáticos de problemas reales.
- Análisis de error y estabilidad numérica.
- Rendimiento. Normalmente no es posible evitarlo...
- Propuestas paralelas... (des)afortunadamente.

De un Call for Papers

"The use of supercomputing technology, parallel and distributed processing, and sophisticated algorithms is of major importance for computational scientists. Yet, the scientists' goals are to solve the challenging problems, not the software engineering tasks associated with it. For that reason, computational science and engineering must be able to rely on dedicated support from program development and analysis tools. Focusing on this background, the following question must be investigated:

How to support users of computational science and engineering during program development and analysis?"

Desde la perspectiva de un curso básico de cómputo paralelo en clusters:

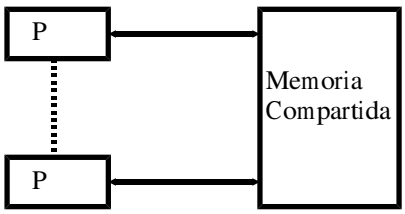
- Los algoritmos paralelos no son incialmente pensados o propuestos para clusters.
- Los clusters tienen diferencias muy marcadas con las computadoras paralelas *tradicionales* (¿?).
- La mayoría de los algoritmos deberían ser analizados desde la perspectiva del cómputo en clusters.

2 Hardware de Procesamiento Actual

Clasificación de Flynn como referencia: secuencias (*streams*) de datos y de instrucciones: SISD, SIMD, MISD, MIMD: memoria compartida y memoria distribuida. Multiprocesadores y multicomputadoras son las computadoras paralelas *tradicionales*.

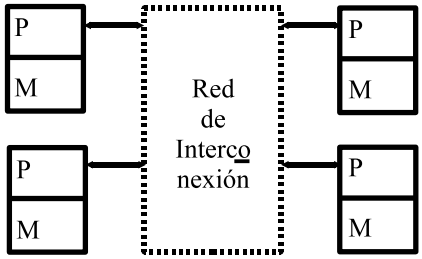
Dos clases de MIMD

- Memoria compartida (multiprocesadores)



  - Máquina única-muchos procesadores
  - Una única visión de la memoria (mapa de memoria)

- Memoria distribuida (multicomputadoras)



  - DMPC: lo mismo
  - Multicomputadora: muchas computadoras. Máquina única-muchos P+M
  - Acoplamiento débil: independencia, asincronismo
  - Modelo de programación: Pasaje de mensajes, CSP

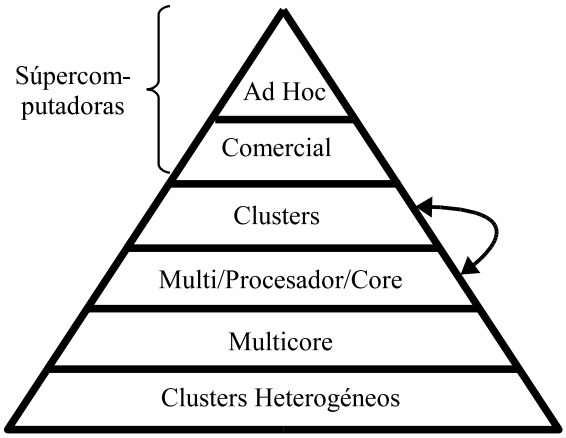
Clasificación de Flynn: SISD - SIMD - MISD - MIMD

- ¿Pipeline? ¿Superescalares? ¿Arquitecturas Harvard?
- ¿Por qué se considera que MIMD es la más general? Aplicable a una amplia gama de problemas (al menos más amplia) que las demás arquitecturas
- ¿Por qué se considera que MIMD es la más escalable? Escalabilidad: capacidad de aumentar la cantidad de recursos para resolver problemas mayores (en datos y/o en procesamiento) "Más escalable": no necesita sincronismo al nivel de clock

Empuje del mercado/velocidad/MHz (más o menos cronológico):

- Más MHz: tecnología.
- Más MHz: pipelining (*segmentación de cauce*).
- Diferencias de velocidad CPU/memoria: jerarquía de memoria.
- Más recursos para algunas operaciones: procesadores súperescalares.
- Mejor ocupación de los recursos: ejecución especulativa.
- Mejor ocupación de los recursos: múltiples hilos (threads).
- Tecnología, consumo/disipación, cache/s,...: múltiples núcleos.
- *Otros* recursos: GPUs.

Otra clasificación más:



¿"Many cores" (decenas o centenas de núcleos)?

Práctica con Multiplicación de Matrices Secuencial

Evaluación de rendimiento secuencial

Multiplicación de Matrices. Dadas

$$A \in I\!R^{m \times k};\ a_{ij}, 1 \leq i \leq m, 1 \leq j \leq k$$

y

$$B \in I\!R^{k \times n};\ b_{ij}, 1 \leq i \leq k, 1 \leq j \leq n$$

El resultado es la matriz

$$C = A \times B;\ C \in I\!R^{m \times n};\ c_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$$

que se obtiene con

$$c_{ij} = \sum_{r=1}^{k} a_{ir} b_{rj}$$

A) Para hacer en 10 (o 15) minutos:
- A.1) Cantidad de operaciones que se requieren.
- A.2) Cantidad de operaciones de la implementación clásica.

B) Con la implementación clásica de las tres iteraciones:
- B.1) Las tres iteraciones. Tiempos para $n = 100$, $n = 200$, ... $n = 500$, $n = 1000$...
- B.2) Las optimizaciones del compilador. Los tiempos.
- B.3) Qué es lo que se puede entender/explicar.
- B.4) ¿Es lo que se esperaba de rendimiento? ¿Cómo medir rendimiento?
- B.5) ¿Cuántas multiplicaciones con $n = 100$ serían equivalentes a una con $n = 1000$?
- B.6) ¿Podría ser más rápido? ¿Por qué es importante?

C) Optimizaciones orientadas al aprovechamiento de la jerarquía de memoria:
- C.1) La idea del procesamiento por bloques: un elemento, un bloque, ...
- C.2) Las ideas de los cursos de cómputo de alto rendimiento/paralelo.

D) Con la implementación con bloques:
- D.1) En 10 (o 15) minutos: ¿Cuál es el que está implementado?
- D.2) El rendimiento, tiempo.
- D.3) ¿Es lo que se esperaba de rendimiento? ¿Cómo medir rendimiento?
- D.4) La equivalencia de $n = 100$ con $n = 1000$.

E) Con la biblioteca optimizada...
- E.1) El rendimiento, tiempo.
- E.2) ¿Es lo que se esperaba de rendimiento? ¿Cómo medir rendimiento?
- E.3) La equivalencia de $n = 100$ con $n = 1000$.

Breve comentario/ejemplo de procesamiento por bloques
Específicamente aplicado a la multiplicación de matrices de 4×4: A, B y C se subdividen en bloques de 2×2 ($bs = 2$). Cada bloque/submatriz tendrá sus correspondientes índices de fila y columna

C_{00}		C_{01}	
c_{00}	c_{01}	c_{02}	c_{03}
c_{10}	c_{11}	c_{12}	c_{13}
c_{20}	c_{21}	c_{22}	c_{23}
c_{30}	c_{31}	c_{32}	c_{33}
C_{10}		C_{11}	

$=$

A_{00}		A_{01}	
a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}
A_{10}		A_{11}	

x

B_{00}		B_{01}	
b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}
B_{10}		B_{11}	

Resultado Parcial: $C0_{00} = A_{00}$ x B_{00}
Resultado Parcial: $C1_{00} = A_{01}$ x B_{10}

$$C0_{00} = \begin{array}{|cc|}\hline a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \\ \hline\end{array}$$

$$C1_{00} = \begin{array}{|cc|}\hline a_{02}b_{20} + a_{03}b_{30} & a_{02}b_{21} + a_{03}b_{31} \\ a_{12}b_{20} + a_{13}b_{30} & a_{12}b_{21} + a_{13}b_{31} \\ \hline\end{array}$$

Por lo tanto, $C0_{00} + C1_{00}$ debería ser C_{00}, es decir los cuatro elementos del *rincón izquierdo superior*...

$$\begin{array}{|cc|}\hline a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} + a_{03}b_{30} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} + a_{03}b_{31} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} + a_{13}b_{30} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ \hline\end{array}$$

Es análogo para los demás bloques de C y se puede demostrar la validez de la idea en general, para cualquier n y bs.

Breve comentario/ejemplo de compiladores, código fuente y rendimiento
Código no optimizado, en plataformas de 32 y 64 bits (Linux FC6), el binario generado y el rendimiento obtenido depende muy fuertemente del compilador y de las opciones de compilación utilizadas. Ejemplo: programa de simulación climática en un AMD Athlon 64 3000+, 1.8 GHz

Binario	**ifort 9.0**	**ifort 10.0**	**Dif. t**
32 bits	01 h. : 37 min.	00 h. : 50 min.	47 min.
64 bits	00 h. : 52 min.	00 h. : 34 min.	18 min.
Dif. t	45 min.	16 min.	

3 Hardware MIMD de Procesamiento Paralelo

MIMD - Multiprocesadores

1. Con toda la memoria compartida y en un solo bloque
   - Ventajas: Sincronización por acceso a memoria, Comunicación de procesos por acceso a memoria, "historia" de la concurrencia: (Conc. - par.).
   - Desventajas: t de acceso a memoria (ya es un problema con 1 procesador), Accesos simultáneos a memoria: Memoria en bancos, buses.
   - Tiempo de acceso a memoria: $t_{mem} + t_{col(frec, \#proc)}$.
2. Agregando Cache a los procesadores
   - Reducir los requerimientos de acceso a memoria: frecuencia de accesos a la memoria compartida.
   - Problema: Falta de coherencia de memorias cache $\Longrightarrow$ Hardware (tiempo y transparencia): protocolos de coherencia de cache (snoop o aviso).
   - Tiempo de acceso: $t_{cache} + t_{mem} + t_{col(frec(cache), \#proc)}$.
   - SMP: caso particular, énfasis en el acceso a todos los recursos.
3. Agregando memoria local independiente de memoria compartida ("local data")
   - Ventaja: independencia de accesos a la memoria local.
   - Desventajas: Hardware-"discriminación" de accesos a memoria.
   - Tiempo de acceso: $t_{cache} + t_{meml} + t_{mem} + t_{col(frec(cache,ml), \#proc)}$.
4. Memoria compartida físicamente distribuida
   - SGI Origin.
   - La visión de la memoria sigue siendo única.
   - Sigue habiendo problemas con coherencia de caches.
   - Tiempo de acceso: $t_{cache} + t_{meml} + t_{memr} + t_{col(frec(ml,mr), \#proc)}$.

Las últimas dos son NUMA por construcción, por su misma arquitectura.

MIMD - Multicomputadoras

- La red de interconexión comunica computadoras o procesadores, no memoria.
- La comunicación entre procesadores: I/O, y de hecho no es acceso a memoria.
- Los bloques de P/M son "convencionales", las mayores variaciones se dan en:
  - Canales o links: hardware (a veces en el procesador) para interconexión.
    - Ej1: transputers: diseñados explícitamente con canales.
    - Ej2: DSP (Digital Signal Processors): con varios ports de I/O y canales de DMA más la documentación necesaria para utilizarlos.
  - Redes de interconexión de procesadores (entre los procesadores)
    - Estáticas: interconexiones fijas entre procesadores, vecindario.
    - Dinámicas: Conexiones pto. a pto. pueden variar en el tiempo.
- Caso muy particular de MIMD débilmente acoplado: clusters
  - Algo más o menos nuevo.
  - Computadoras conectadas a una red local (para HPC o estándar).

Programación de Multiprocesadores

Aprendiendo OpenMP

Sitio de OpenMP con las especificaciones y documentación
http://www.openmp.org

El tutorial clásico, que en realidad es más un documento de referencia, excepto en el principio, donde se dan algunas características generales.
https://computing.llnl.gov/tutorials/openMP/

Mientras se utilicen solamente directivas al compilador, el código se puede compilar sin OpenMP y genera un ejecutable con procesamiento secuencial. Las funciones de biblioteca ya introducen cambios en el código que deben quitarse para que sea posible luego generar el binario sin OpenMP.

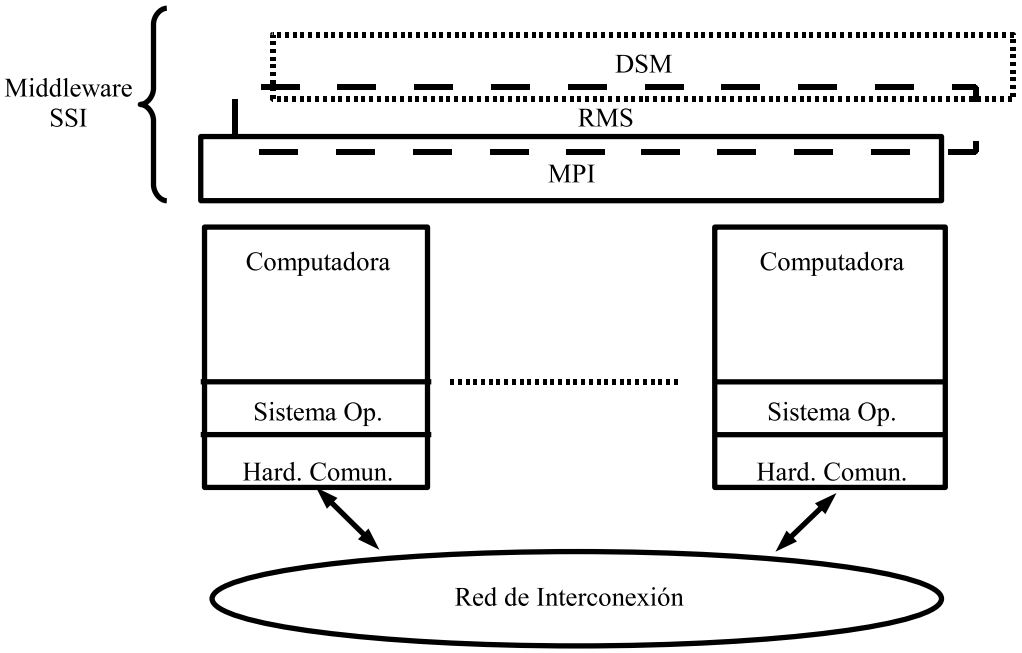
A) Para hacer en 10 (o 15) minutos con el programa hello.c:
- A.1) Agregar
  `#pragma omp parallel`
  antes del primer printf() ¿Explicación? En la línea de comandos:
  `export OMP_NUM_THREADS=7`
  ¿Explicación? Hacer el cambio necesario para que se ejecute todo en hilos.
- A.2) Incluir una llamada a sleep() en la línea anterior al incremento de i.
- A.3) Incluir lo necesario para evitar condiciones de carrera (race conditions).

B) Con el programa de multiplicación de matrices:
- B.1) Utilizar `#pragma omp for` ¿Habría alguna alternativa?
- B.2) Experimentar con diferentes `for` a paralelizar. Explicar cada una.
- B.3) Explicar el rendimiento obtenido.

C) En general, para paralelizar con OpenMP:
- C.1) Dar una secuencia sencilla de pasos a seguir.
- C.2) ¿Sería posible seguir esa secuencia para todas las aplicaciones?
- C.3) ¿Cuáles serían las características más problemáticas para el rendimiento?

4 Clusters como Computadoras Paralelas

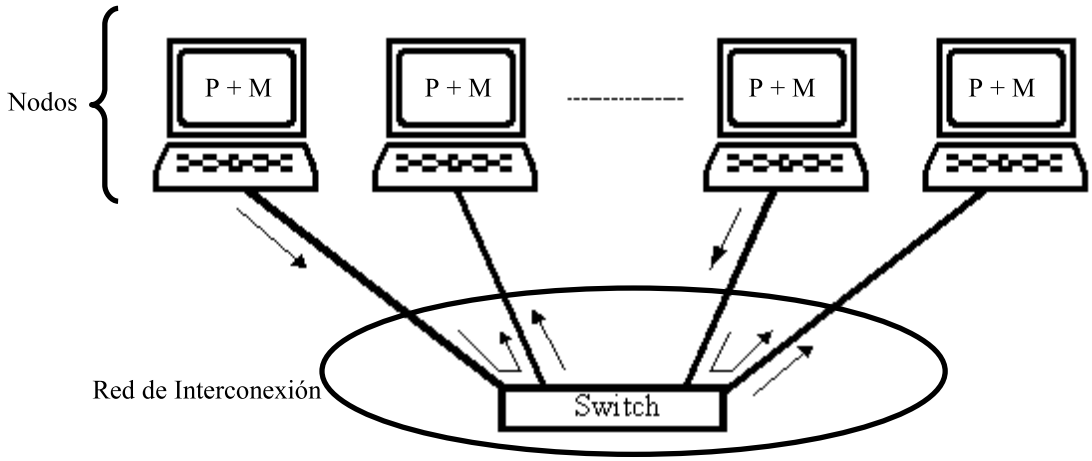
Caracterización y terminología genérica: Capas - Visiones



Características de Rendimiento

- Los clusters no nacen como máquinas paralelas.
- Las CPUs puede ser clasificadas como de "High Performance".
- Desfasaje entre Cómputo y Comunicación.
- LAN, WAN, MAN, SAN, diferentes objetivos.
- Gran énfasis actual en redes "ad hoc" (costo).
- Algoritmos: áreas nuevas, "trasladados".
- Caracterización de rendimiento *aceptable*...
- Cada capa agrega su overhead.
- Algunas capas no se pensaron para procesamiento paralelo.
- Algunas capas para procesamiento paralelo tienen más overhead del *aceptable*.
- Las capas dan una visión, no rendimiento.
- Overhead $\Longrightarrow$ Aumentar Granularidad.
- Heterogeneidad $\Longrightarrow$ Aplicaciones "aware".

Hardware de los clusters: computadoras (PCs y/o estaciones de trabajo) y red de interconexión. Red de interconexión: LAN, Ethernet, Infiniband, etc. (recordar el esquema de una multicomputadora).



4.1 Utilización de Clusters

Tres (¿Cuatro?) Grandes Areas:

- HPC (High Performance Computing)/Aplicaciones.
- Productividad (High Throughput) ¿Paralelo?
- Server Farms ¿Paralelo?
- ¿Desarrollo?

1.- HPC:

- Motivación
  - Tienen todo: CPUs y comunicación entre ellas.
  - Costo/rendimiento (costo por Mflop/s).
- Pasaje de mensajes disponible
  - Sockets.
  - MPI (portable), previamente PVM.
  - Otros.
- Muchas aplicaciones ya hechas
  - Estabilidad (sucess stories).
  - Reusabilidad.
  - Diferentes clases-áreas.

1.1 HPC - Problemas:

- No todos los problemas resueltos.
- No todos los algoritmos son "usables".
- No se conoce el grado de granularidad (imprecisa, *a priori* muy gruesa).
- Aún no hay estabilidad en desarrollo.
- Aún no hay estabilidad en debugging.
- Aún hay más "success stories" que cosas estables.

2.- Productividad (High Throughput):

- RMS (Resource Management and Scheduling).
- Motivación: CPUs disponibles (M. Livny, University of Wisconsin-Madison)
- Requerimientos
  - Computadoras en red - Instituciones
  - Identificación de carga de trabajo
  - Mecanismo de ejecución remota
  - Mecanismo de monitorización
  - Mantenimiento de colas de trabajos (batch)
  - Mecanismo de cancelación o migración

2.1 Productividad - Funcionamiento (CONDOR):

- Manejador de colas + utilización de ciclos libres
- Red local monitoreada (procesamiento)
- Identificación de carga (libre o con usuario)
- Disparo de procesos en computadoras libres
- Migración de procesos a computadoras libres
- Acceso a archivos remotos abiertos
- Aplicaciones "linkeadas" y "no linkeadas"
- Manejador de colas

2.2 Productividad - ¿Paralelo?

- Para el que lo hace quizás lo sea
- Los trabajos que corren no necesariamente...
- ¿Gang-Coscheduling?
- Manejadores de colas $\Longrightarrow$ CPU intensivos
- Carga en la red... "sucess stories"

3.- Desarrollo:

- Aún en las universidades
- "Ejemplo Clementina"
- Clusters de Producción y de Desarrollo
- Implica
  - No escribir sobre los de producción
  - Prioridad para desarrollo... inversión
  - Debugging
  - Evaluación de rendimiento
  - Monitorización-Sintonización
- Trabajos

4.2 Modelos de Programación y de Procesamiento

- Modelos de programación para cómputo paralelo en MIMD
  - Memoria compartida: procesos o hilos.
  - Pasaje de mensajes: procesos, CSP.
- Modelo de procesamiento paralelo
  - SPMD.
  - MPMD.
  - Master/Slave.
  - Otro/s.

Programación de Multicomputadoras

Aprendiendo MPI

Podría considerarse el sucesor estándar de PVM (Parallel Virtual Machine), que es considerado ahora *obsoleto*, aunque hasta el 2009 se hizo el congreso Euro PVM/MPI.

Programa paralelo con MPI: conjunto de procesos independientes para los cuales MPI provee identificación única de procesos y rutinas de comunicaciones.

MPI fue definido para proveer un modelo de programación estándar para multicomputadoras y que puede usarse también en multiprocesadores (¿?). "The official MPI (Message Passing Interface) standards documents, errata, and archives" are found in the "Message Passing Interface (MPI) Forum Home" site http://www.mpi-forum.org/

El estándar MPI en sus versiones 1.0 y 1.1 define un conjunto bastante grande (o muy grande) de rutinas de comunicaciones entre procesos que puede ser relacionada con las definiciones de CSP. Ya la versión 2.0 incluye ideas no relacionadas con CSP, tales como acceso a memoria remota y manejo de procesos (PVM). El entorno de ejecución no es parte de la definición de MPI.

En los clusters se utiliza alguna de las implementaciones disponibles para implementar y ejecutar aplicaciones de alto rendimiento. Entre las más conocidas:

- MPICH: implementada directamente sobre TCP/IP, pero en principio puede cambiarse el dispositivo (*device-port*). En principio era la más sólida, aunque ahora no necesariamente es la mejor.
- OpenMPI: lamentablemente confunde un poco el nombre con OpenMP... Tuvo como *predecesora* a LAM/MPI.
- Todas las empresas de hardware de cómputo paralelo tienden a proveer una implementación de MPI. Sun: Sun Cluster Tools, Intel: Intel MPI, etc.
- Todas las empresas de hardware de red para cómputo paralelo en clusters además suele proveer su propia implementación de MPI. Myricom-Myrinet: MPICH-MX, Infiniband: MVAPICH, etc.
- Las empresas suelen utilizar la versión MPICH y cambiar el *device-port*.

Otro tutorial clásico, que también es útil como material de referencia y con explicaciones muy claras sobre el principio. Es un poco "más tutorial" que el de OpenMP, aunque también se puede usar como material de referencia.

https://computing.llnl.gov/tutorials/mpi/

Cualquier programa con MPI tiene uso explíto de funciones de la biblioteca, es decir que se hace directamente para procesamiento paralelo.

A) Para hacer en 10 (o 15) minutos con el programa mpifirst.c:
   A.1) Compilar y ejecutar (ver `tocomm` y `toexec`).
   A.2) Verificar diferentes asignaciones de procesos a nodos del cluster.
   A.3) Agregar otras funciones de MPI para conocer detalles del ambiente de ejecución.
   A.4) ¿Es posible averiguar cuál es el núcleo sobre el que se está ejecutando un proceso?

B) Para hacer en 10 (o 15) minutos con el programa mpisecond.c:
   B.1) Comunicar otros tipos de datos y arreglos unidimensionales y bidimensionales.
   B.2) ¿Se podrían intercambiar send() y recv()?
   B.3) ¿Habría algún inconveniente en comunicar bloques de matrices?

C) Probar otras alternativas de comunicaciones:
   C.1) Comunicaciones colectivas: `MPI_Bcast()`.
   C.2) Envío inmediato: `MPI_ISend()`.

5 Más de rendimiento

- Rendimiento de los nodos
  - Secuencial: muy bueno (¿siempre?).
  - Paralelo: multiprocesadores.
  - Relativo: homogéneo y heterogéneo.
  - Recordar que en un nodo de la figura en realidad ya hay cómputo paralelo.
- Rendimiento de la red de interconexión
  - Latencia y ancho de banda.
  - Ethernet: basado en bus (CSMA/CD), "dominio de colisión".
  - Ethernet + switch no reduce la latencia.
  - Otras redes...

5.1 Rendimiento de Cómputo de los Nodos

Nodos multiprocesadores: variantes y paralelización.

Nodos homogéneos $\Longrightarrow$ no hay muchos problemas "nuevos".

Nodos heterogéneos $\Longrightarrow$ dos tareas agregadas:

- Métricas de rendimiento específicas.
- Balance de carga no inmediato.

Métricas de rendimiento específicas

1) Programas específicos: el mismo procesamiento pero escalado a un ¿procesador o nodo? Esto no significa utilizar el programa paralelo con una sola tarea.

2) Benchmark generales: generalizaciones $\Longrightarrow$?

Algebra lineal: dados p nodos (computadoras), $comp_i$, $0 \leq i \leq p-1$, con la potencia de cómputo de cada uno con $Mflop/s(comp_i)$, la potencia de cómputo relativa, $rp(comp_i)$, se puede calcular con

$$rp(comp_i) = rp_i = \frac{Mflop/s(comp_i)}{\max_{j=0...p-1}(Mflop/s(comp_j))}$$

Tanto la utilización como la idea subyacente de esta métrica es similar utilizando las funciones min() o avg() en vez de max(). Sin embargo, puede ser preferible utilizar directamente la potencia de cómputo normalizada, $np(comp_i)$, que puede ser definida como

$$np(comp_i) = np_i = \frac{Mflop/s(comp_i)}{\sum_{j=0}^{p-1}(Mflop/s(comp_j))}$$

donde

$$0 < np_i < 1$$
$$\sum_{i=0}^{p-1}(np_i) = 1$$

Balance de Carga Computacional

Si no es posible alguna forma "automática" (master/slave), se puede pensar en balance de carga dinánico y estático. Balance dinámico se justifica cuando no hay conocimiento a priori del procesamiento. Estático es el considerado de mejor rendimiento para operaciones de álgebra lineal.

Cantidades de flops, balance de carga estático y np_i tal como está definida:

¿Procesamiento por bloques? (cantidad de flops corresponde al método).

¿Métodos iterativos?

5.2 Rendimiento de la Red de Interconexión

Indices estándares: latencia (*startup*) y ancho de banda (*bandwidth*) o tasa de transferencia. Modelo de tiempo de un mensaje:

$$t(n) = \alpha + \beta n$$

donde n es la cantidad de datos, α es la latencia y β es $bndw^{-1}$ con $bndw$ el ancho de banda en cantidad de datos por unidad de tiempo. Usualmente, latencia y ancho de banda son calculados experimentalmente. Si no se toma en cuenta la latencia, o la cantidad de datos es suficientemente grande, el costo de tiempo total de un mensaje se puede estimar directamente con

$$t(n) = \beta n$$

que tiene muy poco error cuando $\alpha \ll \beta n$.

Con estas definiciones (suposiciones), ya se estaría en condiciones de estimar experimentalmente α y β...

En general, es más sencillo mejorar el ancho de banda que la latencia...

Granularidad: Rendimiento de Cómputo y Comunicaciones

Experimentación: Relación entre las Métricas

A) En una computadora:

A.1) Determine los Mflop/s.

A.2) Si la latencia es de 0.5 ms ¿cuántas operaciones de punto flotante se podrían hacer como mínimo por cada operación de comunicaciones?

B) Asumiendo que tiene un cluster de computadoras como las del punto anterior con la latencia de comunicaciones también dada en el punto anterior. Si se hacen los cálculos de multiplicación de matrices en bloques ¿qué tamaño propondría para el bloque y cómo modelizaría el rendimiento esperado?

C) En el contexto de la multiplicación de matrices:

C.1) ¿Por qué se suele asumir que para algún tamaño de n el tiempo de comunicaciones puede llegar a ser mucho menor que el de cómputo?

C.2) ¿Hay forma de acotar o caracterizar la suposición anterior?

D) Desarrolle un programa (o dos) para que experimentalmente estime α y β. Explique los recaudos que tomaría para intentar acotar el error de las estimaciones.

E) Con lo desarrollado en el punto anterior:

E.1) ¿Podría estimar α y β en un multiprocesador?

E.2) ¿Sería útil la estimación de α y β en un multiprocesador?

6 Algoritmos Específicos para Operaciones Simples

Primero: no hay operaciones complejas. La más simple, por supuesto, es la multiplicación de matrices. Lo que ya se tiene bien definido y conocido es que $flopsMM$ es

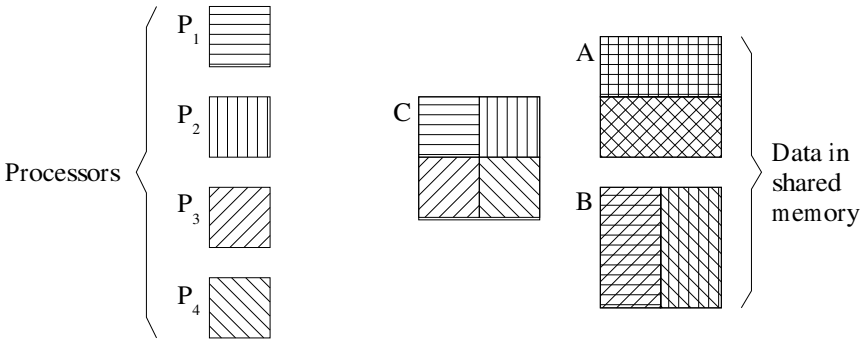
$$flopsMM = 2n^3 - n^2$$

para matrices cuadradas de orden n. Los algoritmos para multiplicación de matrices en general se podrín clasificar en algoritmos para:

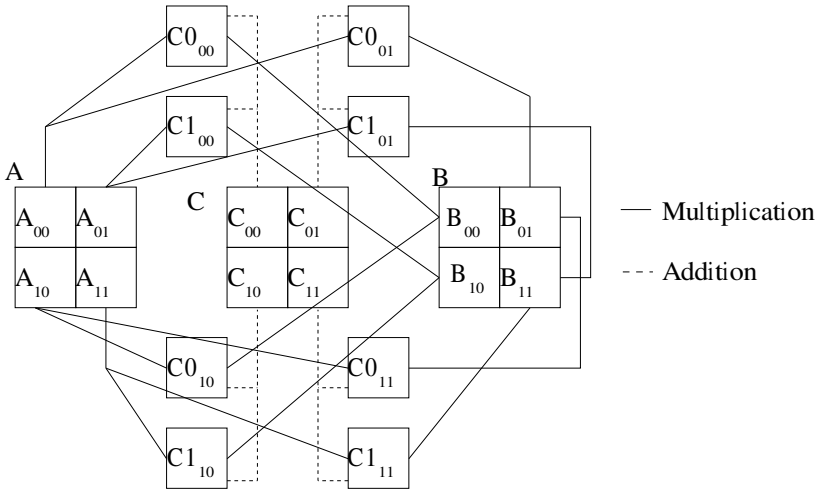
- Multiprocesadores (MIMD de memoria compartida).
- Multicomputadoras (MIMD de memoria distribuida).
- Clusters

6.1 Multiprocesadores

El más sencillo



Con recursión



```
mat_mul(A, B, C, s) /* C = A×B */
/* A, B: operands */
/* C   : result */
/* s   : matrices (square) size */
{
   if (sequential multiplication)
   {
      C = A×B;
   }
   else
   {
      mat_mul(A00, B00, C000, s/2);   /* (1) */
      mat_mul(A01, B10, C100, s/2);   /* (2) */
      mat_mul(A00, B01, C001, s/2);   /* (3) */
      mat_mul(A01, B11, C101, s/2);   /* (4) */
      mat_mul(A10, B00, C010, s/2);   /* (5) */
      mat_mul(A11, B10, C110, s/2);   /* (6) */
      mat_mul(A10, B01, C011, s/2);   /* (7) */
      mat_mul(A11, B11, C111, s/2);   /* (8) */
   }
   C00 = C000 + C100;
   C01 = C001 + C101;
   C10 = C010 + C110;
   C11 = C011 + C111;
}
```

Importante: esto es implícitamente procesamiento por bloques. Para eviar requerimientos de memoria extra se incluye dependencia de datos para los cálculos parciales

```
mat_mul_sum(A, B, C, s) /* C = A×B + C */
/* A, B: operands */
/* C   : result */
/* s   : matrices (square) size */
{
   if (sequential multiplication)
   {
      C = A×B + C;
   }
   else
   {
      mat_mul_sum(A00, B00, C00, s/2);   /* (1) */
      mat_mul_sum(A01, B10, C00, s/2);   /* (2) */
      mat_mul_sum(A00, B01, C01, s/2);   /* (3) */
      mat_mul_sum(A01, B11, C01, s/2);   /* (4) */
      mat_mul_sum(A10, B00, C10, s/2);   /* (5) */
      mat_mul_sum(A11, B10, C10, s/2);   /* (6) */
      mat_mul_sum(A10, B01, C11, s/2);   /* (7) */
      mat_mul_sum(A11, B11, C11, s/2);   /* (8) */
   }
}
```

Método de Strassen: ¿Cómo funciona? ¿Cantidad de flops?

$$P_0 = (A_{00} + A_{11}) \times (B_{00} + B_{11})$$
$$P_1 = (A_{10} + A_{11}) \times B_{00}$$
$$P_2 = A_{00} \times (B_{01} - B_{11})$$
$$P_3 = A_{11} \times (B_{10} - B_{00})$$
$$P_4 = (A_{00} + A_{01}) \times B_{11}$$
$$P_5 = (A_{10} - A_{00}) \times (B_{00} + B_{01})$$
$$P_6 = (A_{01} - A_{11}) \times (B_{10} + B_{11})$$
$$C_{00} = P_0 + P_3 - P_4 + P_6$$
$$C_{01} = P_2 + P_4$$
$$C_{10} = P_1 + P_3$$
$$C_{11} = P_0 + P_3 - P_1 + P_5$$

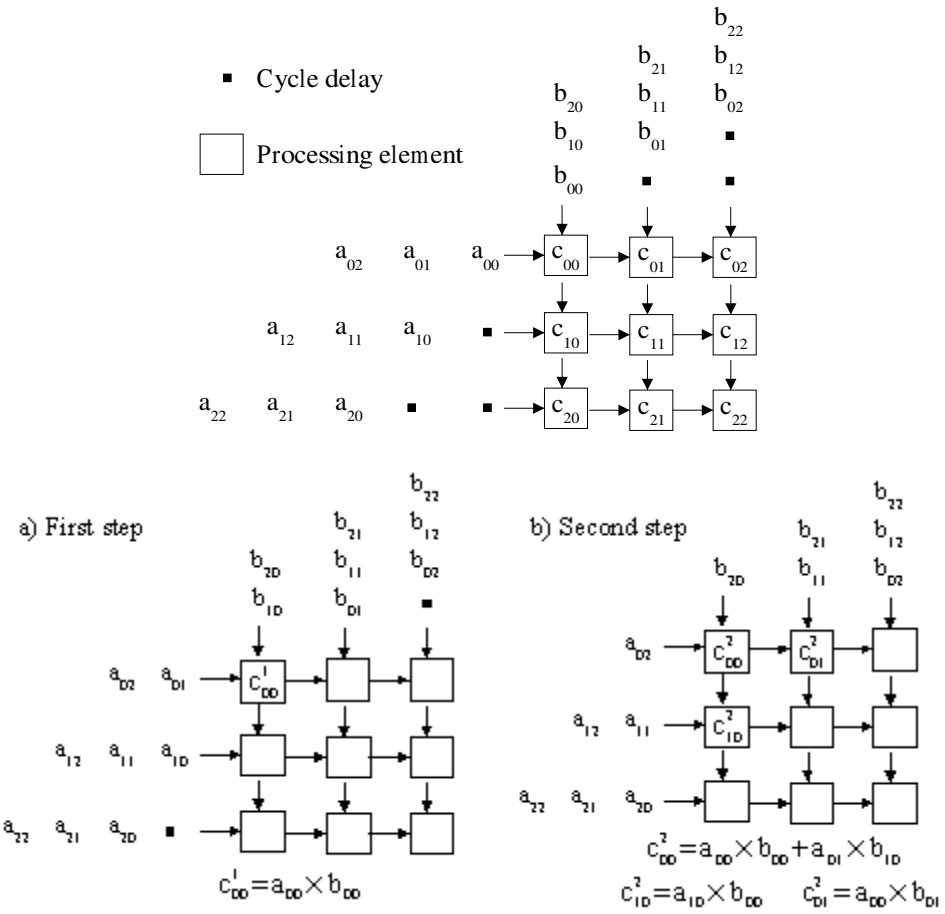
a) Computing with submatrices

$$A = \begin{array}{|c|c|}\hline A_{00} & A_{01} \\ \hline A_{10} & A_{11} \\ \hline \end{array} \quad B = \begin{array}{|c|c|}\hline B_{00} & B_{01} \\ \hline B_{10} & B_{11} \\ \hline \end{array} \quad C = \begin{array}{|c|c|}\hline C_{00} & C_{01} \\ \hline C_{10} & C_{11} \\ \hline \end{array}$$

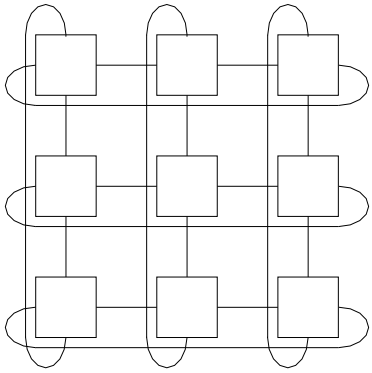
b) Matrix Partition

6.2 Multicomputadoras

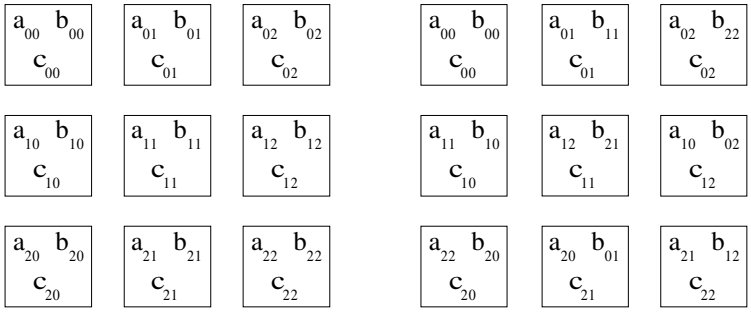
Básicamente Cannon & Fox. Sin embargo,



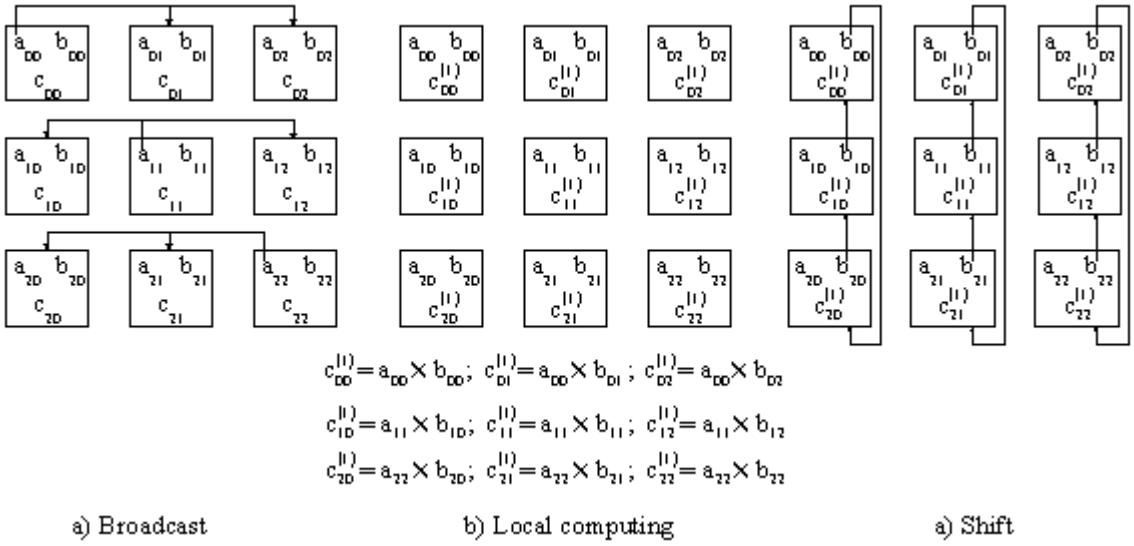
La red de interconexión más comúmnmente tenida en cuenta:

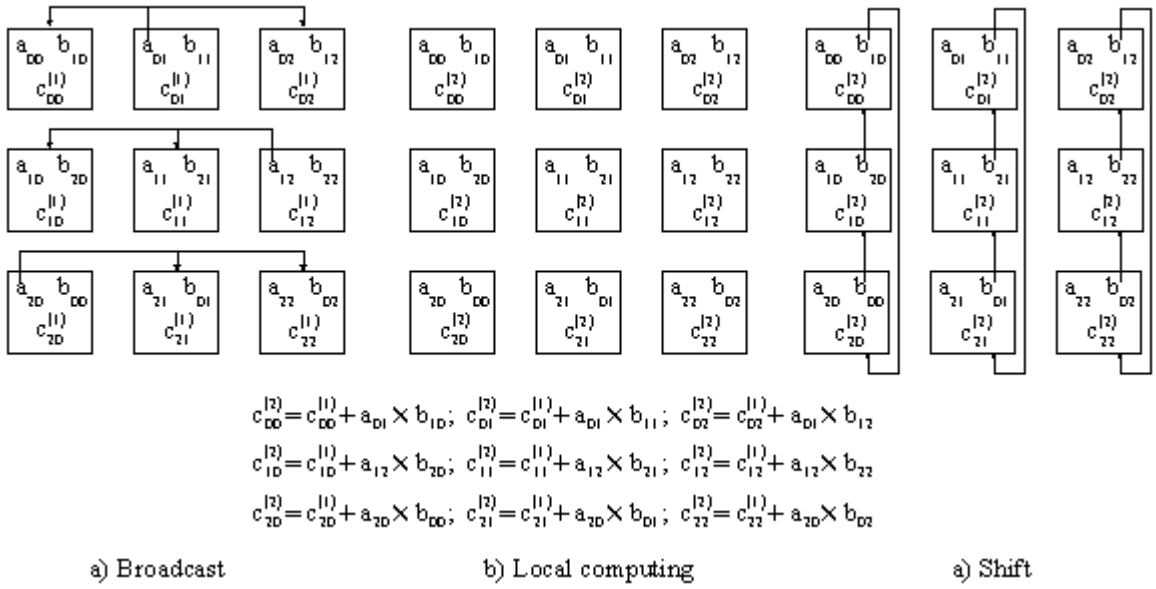


Cannon: reubicación y desplazamientos

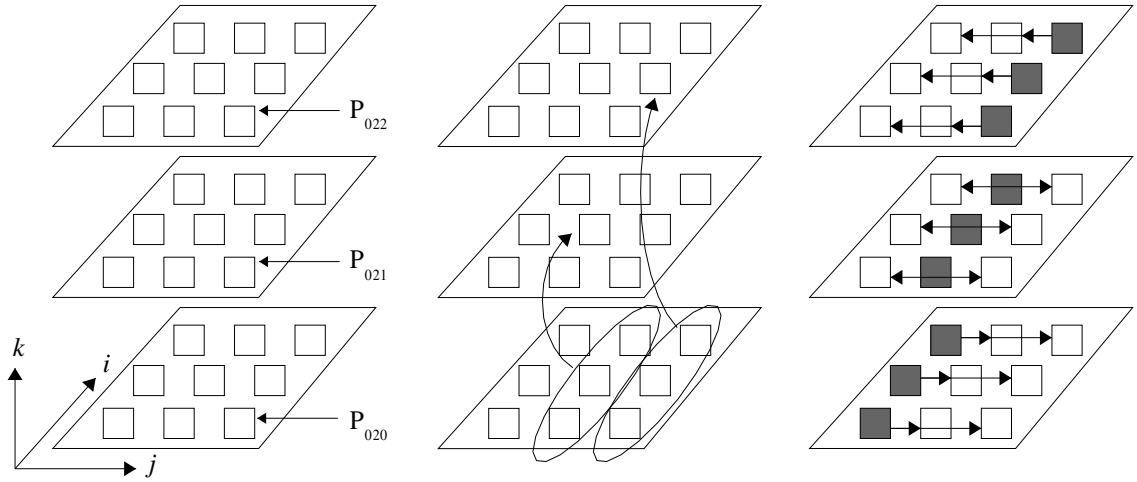


Fox: broadcast y desplazamientos





Arreglos tridimensionales de procesadores: DNS (con replicación de datos)



a) Processors identification. b) Distribution of columns of A. c) Broadcast by Planes.

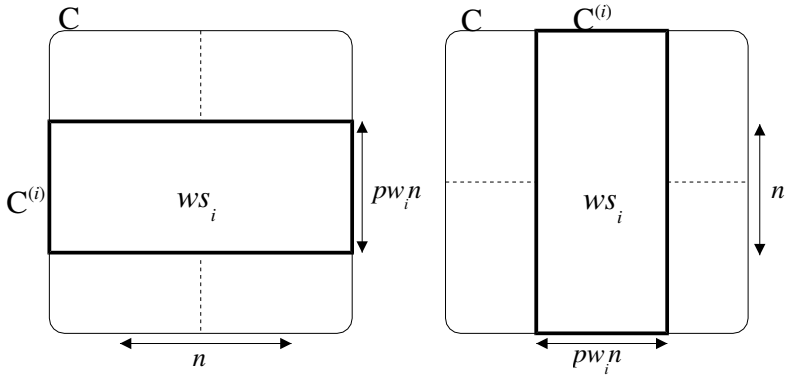
6.3 Clusters

Hay por supuesto mucho trabajo previo en el área de DMPC que se puede reveer, pero directamente orientado a los clusters se tendrán en cuenta la red de interconexión, la heterogeneidad y la simplicidad del algoritmo resultante para definir el algoritmo paralelo.

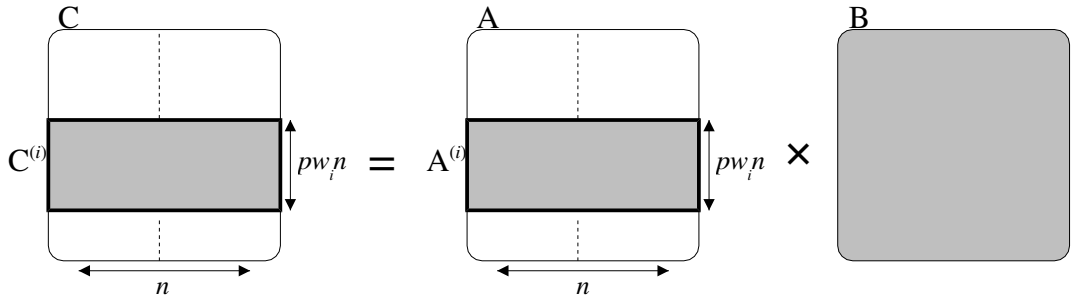
Asumiendo que $pw_i = np_i$ y que np_i está calculado tal como se hizo antes, es decir

$$pw_i = np(comp_i) = np_i = \frac{Mflop/s(comp_i)}{\sum_{j=0}^{p-1}(Mflop/s(comp_j))}$$

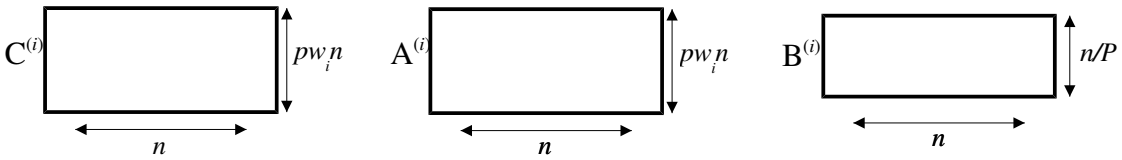
Los cálculos de la matriz C se distribuyen de acuerdo a este pw_i, puede ser por bloques de filas o por bloques de columnas.



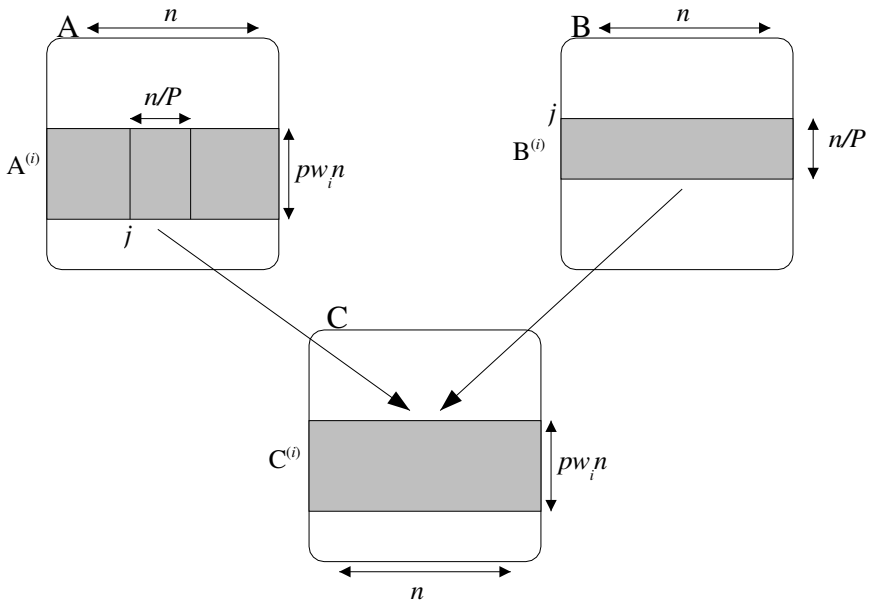
Seleccionando la opción de distribuir por bloques de filas de C queda *automáticamente* definida la distribución de la matriz A por la dependencia de datos que tienen los cálculos



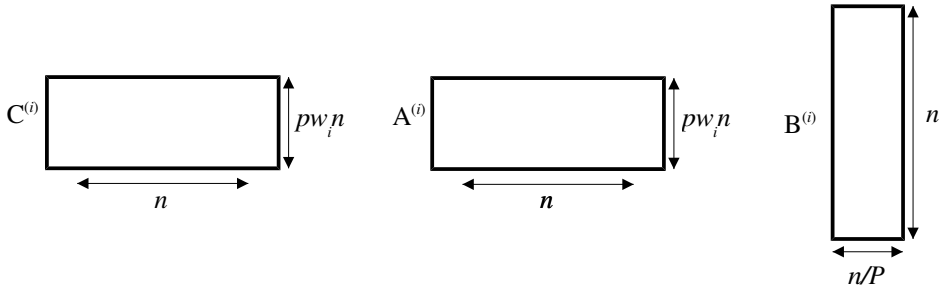
Sin embargo, la matriz B no debería estar replicada. Si se elige distribuir B por bloques de filas



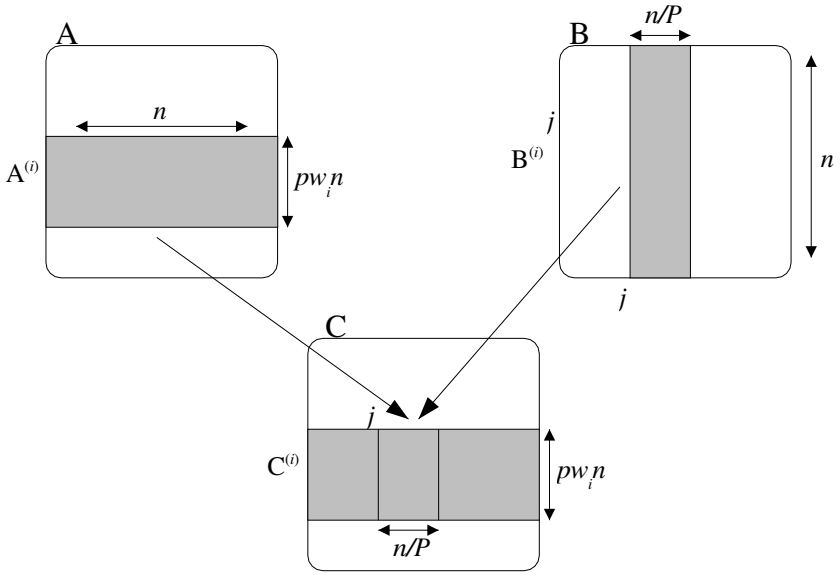
Con los datos de A y B locales, cada nodo podría calcular



Si se elige distribuir B por bloques de columnas

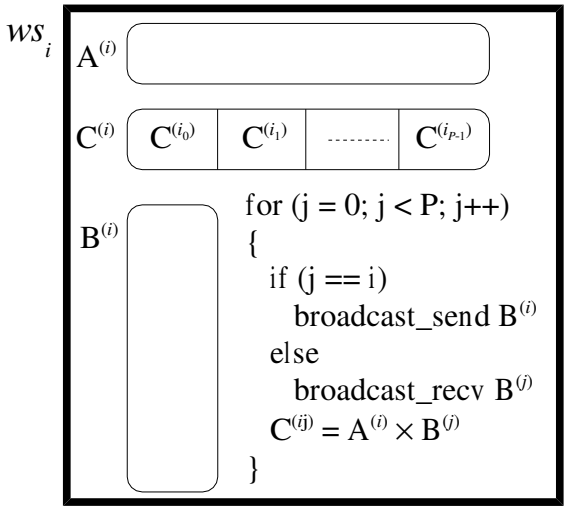


Y con estos datos de A y B locales, cada nodo podría calcular

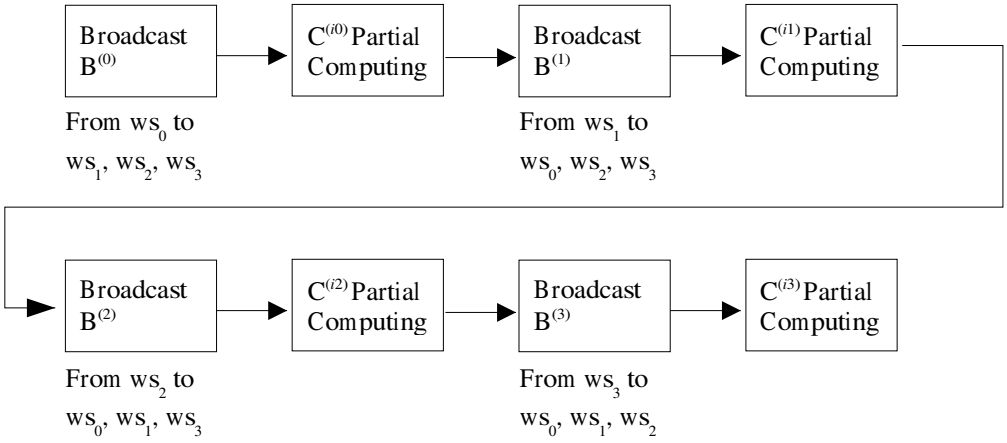


En principio, ninguna de estas opciones para la distribución de la matriz B es mejor que la otra, se elige la distribución por bloques de columnas porque sigue el patrón de la propia definición algebraica de la multiplicación de matrices.

Finalmente, el primer algoritmo paralelo para multiplicar matrices en clusters es



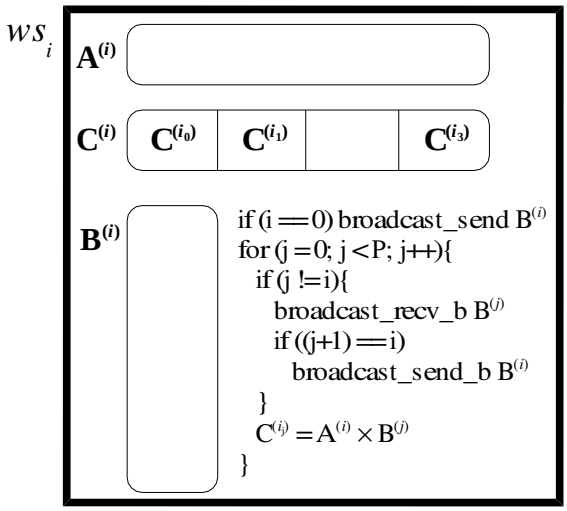
Analizando la secuencia de pasos de cómputo y comunicaciones



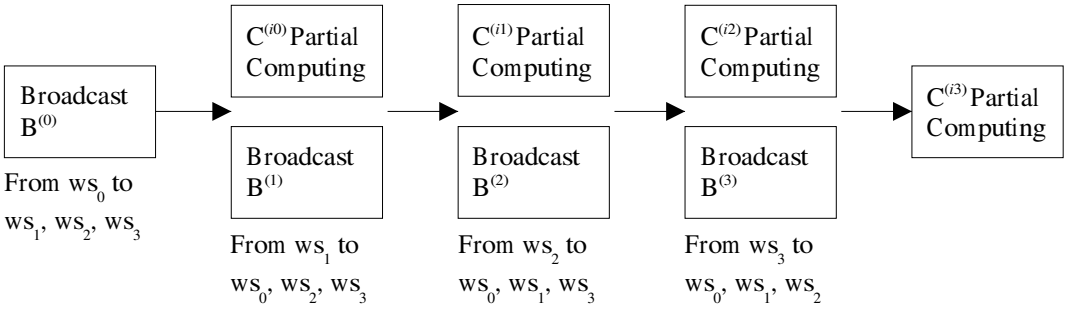
Se debe recordar que toda comunicación significa penalización de rendimiento. Se pueden ver formas (y las hay) para solapar cómputo con comunicaciones. Sin embargo, el algoritmo directamente implementado en Fortran con MPI en un cluster de tres computadoras, cada una con dos dual core AMD Opteron, 4GB RAM, interconectadas con Gb Ethernet y con OpenMPI...

```
$> time myparmm_1_acml 8000 1
 Matrices size:  8000 x 8000  elements
 Number of MPI processes:  1
 Number of OpenMP threads per MPI process:  1
 Memory for data in M(10^6)B:  1024
 Each Bcast is  256 M(10^6)B
real 2m26.632s        user 2m19.932s            sys 0m1.751s

$> time myparmm_1_acml 8000 2
 Matrices size:  8000 x 8000  elements
 Number of MPI processes:  1
 Number of OpenMP threads per MPI process:  2
 Memory for data in M(10^6)B:  1024
 Each Bcast is  256 M(10^6)B
real 1m22.738s        user 2m20.641s            sys 0m1.932s

$> time myparmm_1_acml 8000 3
 Matrices size:  8000 x 8000  elements
 Number of MPI processes:  1
 Number of OpenMP threads per MPI process:  3
 Memory for data in M(10^6)B:  1024
 Each Bcast is  256 M(10^6)B
real 1m1.244s         user 2m21.587s            sys 0m1.971s

$> time myparmm_1_acml 8000 4
 Matrices size:  8000 x 8000  elements
 Number of MPI processes:  1
 Number of OpenMP threads per MPI process:  4
 Memory for data in M(10^6)B:  1024
 Each Bcast is  256 M(10^6)B
real 0m51.280s        user 2m21.566s            sys 0m2.226s

$> time mpirun -np 2 -bynode  -hostfile hosts  myparmm_1_acml 8000 4
 Matrices size:  8000 x 8000  elements
 Number of MPI processes:  2
 Number of OpenMP threads per MPI process:  4
 Memory for data in M(10^6)B:  512
 Each Bcast is  128 M(10^6)B
real 0m27.349s        user 1m10.660s            sys 0m2.502s

$> time mpirun -np 3 -bynode  -hostfile hosts  myparmm_1_acml 8001 4
 Matrices size:  8001 x 8001  elements
 Number of MPI processes:  3
 Number of OpenMP threads per MPI process:  4
 Memory for data in M(10^6)B:  340
 Each Bcast is  84 M(10^6)B
real 0m19.284s        user 0m48.961s            sys 0m2.432s
```

Para solapar cómputo con comunicaciones u *ocultar latencia* de comunicaciones:



Y una ejecución idealizada



Que tiene dos simplificaciones: 1) la comunicación es completamente solapada (como si no se utilizara absolutamente nada de cómputo, cache, etc.), y 2) cómputo y comunicaciones requieren el mismo tiempo. No es tan inmediato de implementar con MPI porque no existen Bcast_send y Bcast_recv, sino solamente MPI_Bcast, aunque no sería tan dificil la implementación.

Siglas/Acrónimos

CSP: Communicating Sequential Processes
DMPC: Distributed Memory Parallel Computer
DSP: Digital Signal Processor
DSM: Distributed Shared Memory
GPU: Graphics Processing Unit
HPC: High Performance Computing
LAN: Local Area Network
MAN: Metropolitan Area Network
MIMD: Multiple Instruction, Multiple Data (streams, de la clasificación de Flynn)
MISD: Multiple Instruction, Single Data (streams, de la clasificación de Flynn)
MPI: Message Passing Interface
MPICH: Implementación de MPI
MPICH-MX: Implementación de MPI de/para Myricom-Myrinet
MPMD: Multiple Program, Multiple Data

MVAPICH: Implementación de MPI de/para Infiniband
NUMA: Non-Uniform Memory Access (multiprocesadores)
PVM: Parallel Virtual Machine
RMS: Resource Management System y/o Resource Management and Scheduling
SAN: Storage Area Network
SIMD: Single Instruction, Multiple Data (streams, de la clasificación de Flynn)
SISD: Single Instruction, Single Data (streams, de la clasificación de Flynn)
SPMD: Single Program, Multiple Data
SSI: Single System Image
UMA: Uniform Memory Access (multiprocesadores)
WAN: Wide Area Network