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

2007

Clusters. Programación en Clusters
Algebra Lineal en Paralelo sobre Clusters

UNLP - 2007

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:

1 Introduction

Programa del Curso publicado

Temas del Curso
    Introducción.
    Algebra lineal: características, aplicaciones, problemas clásicos.
    Evaluación de rendimiento secuencial y paralelo.
    Clusters: arquitectura de procesamiento paralelo.
    Presentación y análisis de bibliotecas de software disponibles.
    Análisis de los problemas y soluciones posibles para el balance de carga.
    Ejemplo de modelización: desde un problema hasta un modelo a paralelizar.
    Análisis de posibles trabajos.
    Levantando el nivel de abstracción:
        Soporte de sistemas operativos
        Bibliotecas en general
        Abstracciones intercluster
        Internet computing
        Grid computing
        ...

Schedule

|	Original	*Current*
Lectures	Every Monday	Every Day, 1 Week
Practice	Written homeworks. Individual. Every Moday. Conclusions.	Some *Little Work*. Individual. Every Day?
Grading	Project. Individual.	Project. Individual.

Objectives:

- Linear algebra.

- Parallel computing on clusters.

- Parallel algorithms analysis (*on clusters*).

- Heterogeneity and workload balance.

- Some practice with MPI.

Assumptions (should be):

- *Minimum* algebra.

- Parallel computing.

- Parallel algorithms analysis.

Too many problems...

Production Software:
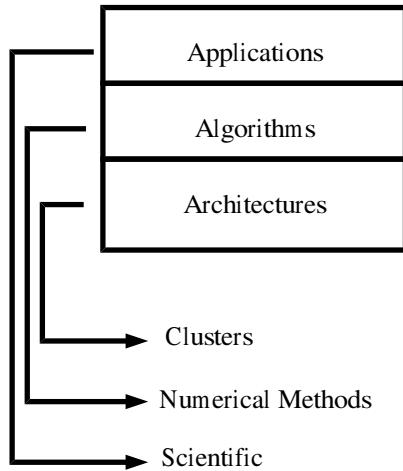
Algorithmic View:
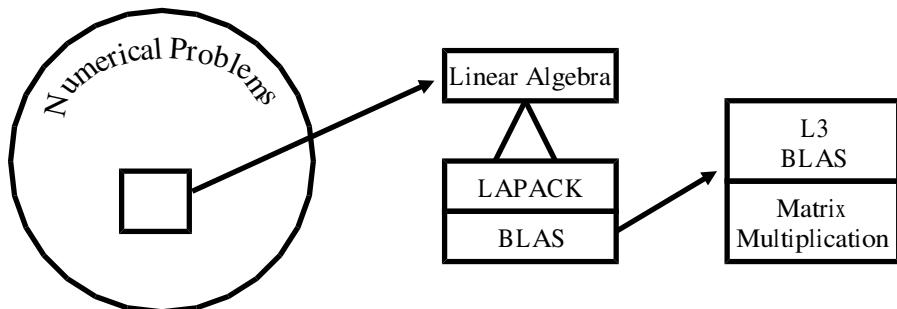
Getting Production Code:
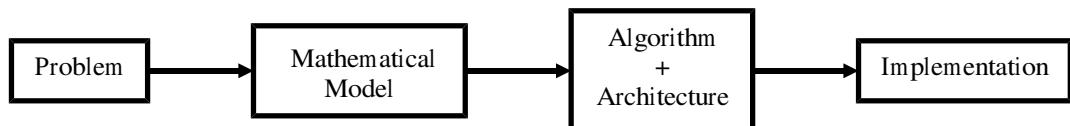
However, Getting Production Code (2):
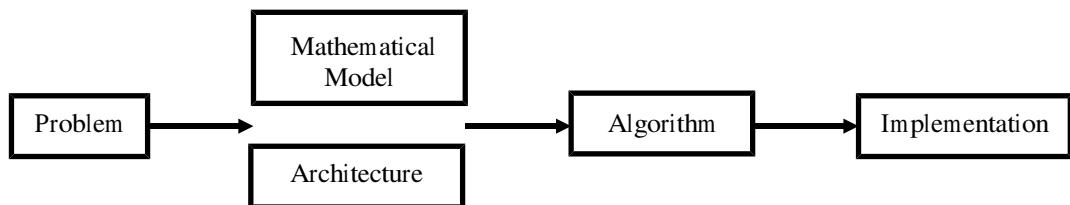
Many algorithms proposed and used with strong emphasis on

- Mathematical models to real problems.

- Numerical stability, error analysis.

- Performance. Well, it is not possible to avoid this...

- Parallel approaches... (un)fortunately.

From a 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?"

From the *Basic parallel cluster computing* ideas:

- Parallel algorithms are not initially proposed for clusters.

- Clusters have strong differences with parallel computers (?).

- Algorithms should be analyzed *in the context of* parallel cluster computing.

1.1   Current Computing Hardware

Just *another* classification:
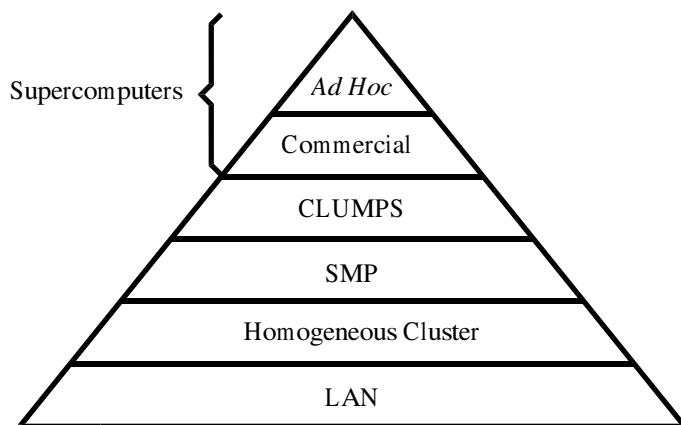
1.2   Utilización de Clusters

Dos (¿Tres?) Grandes Areas:

- High Throughput ¿Paralelo?

- High Performance Applications

- ¿Desarrollo?

- Server Farms ¿Paralelo?

1.- High Throughput:

- Resource Management and Scheduling (RMS)

- Motivación: CPUs disponibles (M. Livny)

- 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

1.1 High Throughput - 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

1.2 High Throughput - ¿Paralelo?

- Para el que lo hace sí lo es

- Los trabajos que corren no necesariamente...

- ¿Gang-Coscheduling?

- Manejadores de colas $\Longrightarrow$ CPU intensivos

- Carga en la red... "sucess stories"

2.- High Performance:

- Motivación

  - Tienen todo: CPUs y comunicación entre ellas
  - Costo/rendimiento (costo por Mflop/s)

- Pasaje de mensajes disponible

  - Sockets
  - PVM, MPI (ambos portables)

- Muchas aplicaciones ya hechas

  - Estabilidad (sucess stories)
  - Reusabilidad
  - Diferentes clases-áreas

2.1 High Performance - Problemas:

- No todos los problemas resueltos

- No todos los algoritmos son "usables"

- No se conoce el grado de granularidad (muy gruesa)

- Aún no hay estabilidad en desarrollo

- Aún no hay estabilidad en debugging

- Aún no hay mucha estabilidad en terminología

- Aún no hay muchas cosas estables, sí hay "success stories"

3.- Desarrollo:

- Aún en las universidades

- "Ejemplo Clementina"

- Clusters de Producción y de Desarrollo

- Implica

    - "No" escribir

    - Debugging

    - Evaluación de rendimiento

    - Monitorización-Sintonización

- Trabajos

2 Linear Algebra

Reasons:

- Applications and users

- Background $\Longrightarrow$ starting point

- Parallel: processing requirements

General characteristics:

- Matrix processing. Matrix Computations, Golub G., C. Van Loan, 2nd Edition, The John Hopkins University Press, 1989.

- Background $\Longrightarrow$ starting point

- (Huge) processing requirements $\Longrightarrow$ parallel approaches

Classical Problems:

- Matrix Multiplication: simple, benchmark, the best

- LU Matrix Factorization: relatively complex, *the benchmark*, *the problem* (applications)

Matrix Multiplication. Given

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

and

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

matrix

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

is obtained by

$$c_{ij} = \sum_{r=1}^{k} a_{ik} b_{kj}$$

LU Matrix Factorization. Given $A \in I\!\!R^{n \times n}$, find out two matrices, L and U such that

- $L \in I\!\!R^{n \times n}$ is a lower triangular, unit diagonal matrix.

- $U \in I\!\!R^{n \times n}$ is an upper triangular matrix.

- $A = L \times U$.

3 A Few Comments on Performance

Evaluación de rendimiento: una vez que *no hay errores...*

¿Cómo funciona?
¿Podría ser más rápido?
¿Por qué es importante?

¿Cómo evaluarían el rendimiento? (no hay nada aquí a propósito, ¿eh?)

Más específicamente:

¿Cómo evaluarían el rendimiento secuencial?
¿Cómo evaluarían el rendimiento paralelo?

4   Parallel Architectures

Clasificación de Flynn: SISD - MISD - SIMD - 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

Dos clases de MIMD

- Memoria compartida (multiprocesadores)

```
 ┌─────┐              ┌──────────┐
 │  P  │──────────────│          │
 └─────┘              │ Memoria  │
    ┊                 │ Compartida│
 ┌─────┐              │          │
 │  P  │──────────────│          │
 └─────┘              └──────────┘
```
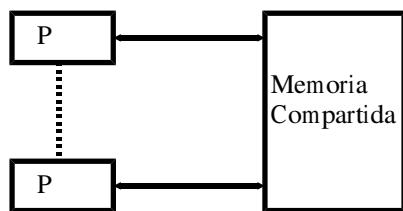
    - Máquina única-muchos procesadores
    - Una única visión de la memoria (mapa de memoria)

- Memoria distribuida (multicomputadoras)

```
┌───┐         ┌──────────┐        ┌───┐
│ P │─────────┊          ┊────────│ P │
├───┤         ┊   Red    ┊        ├───┤
│ M │         ┊    de    ┊        │ M │
└───┘         ┊ Interco  ┊        └───┘
┌───┐         ┊ nexión   ┊        ┌───┐
│ P │─────────┊          ┊────────│ P │
├───┤         ┊          ┊        ├───┤
│ M │         └──────────┘        │ M │
└───┘                             └───┘
```
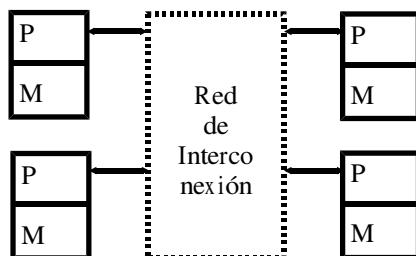
    - DMPC: lo mismo
    - Multicomputadora: muchas computadoras
    - Losely coupled: independencia, asincronismo
    - Pasaje de mensajes: CSP

11

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 losely coupled: clusters

  - Algo más o menos nuevo.
  - Estaciones de trabajo y/o PCs conectadas a una red estándar.

4.1 Clusters como Computadoras Paralelas

Caracterización y terminología: "High Performance Cluster Computing (Architecture, Systems, and Applications)", ISCA-2000, The 27th Annual International Symposium on Computer Architecture, June 10-14, 2000, Vancouver, British Columbia, Canada, Sponsored by ACM SIGARCH, IEEE Computer Society TCCA.

Capas - Visiones
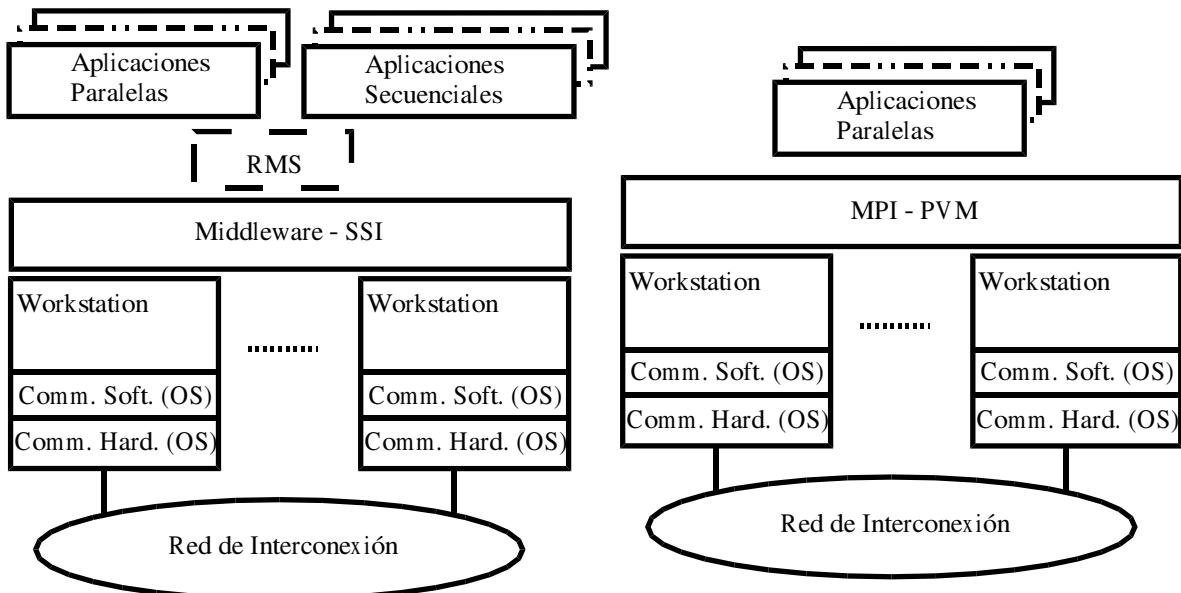
Características de Rendimiento

- Los clusters no nacen como máquinas paralelas.

- Las CPUs suelen ser "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".

4.2 Clusters Parallel Performance

Clusters hardware: computers (PCs and/or workstations) and interconnection network.
Interconnection network: LAN, Ethernet.

Hubs: failures

14
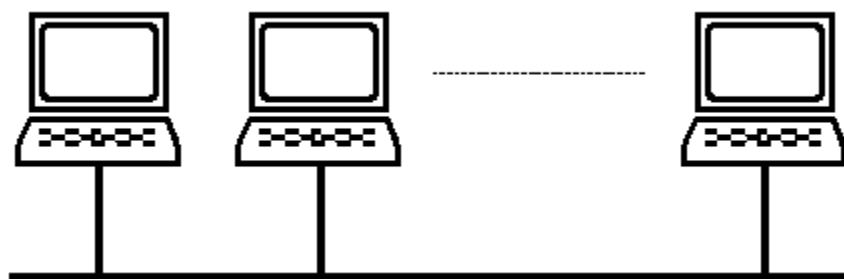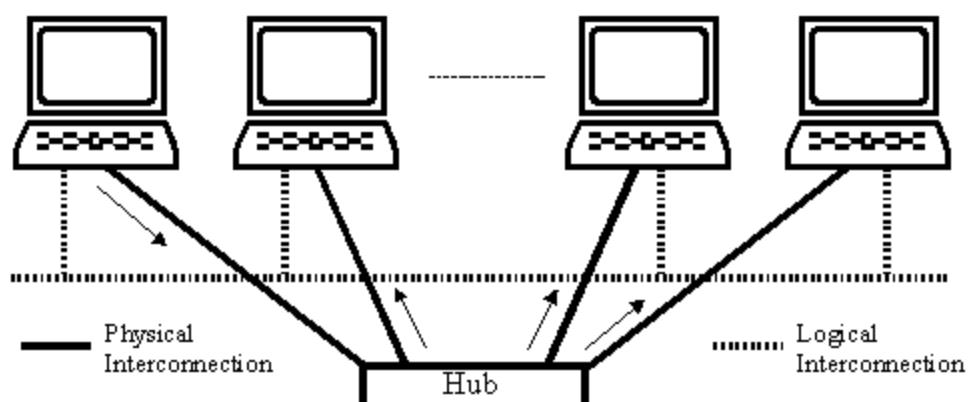
Switches: performance
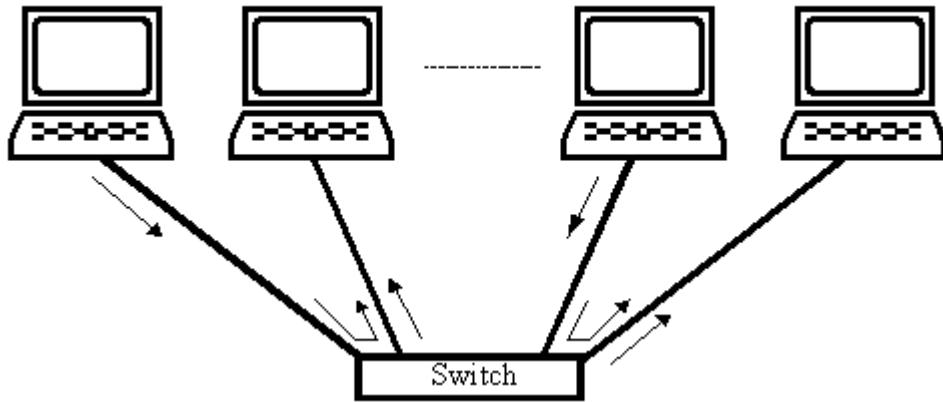
Clusters as parallel computers:

- Computing Nodes Performance

  - Sequential: very good
  - Relative: homogeneous and heterogeneous.

- Interconnection Network Performance

  - Ethernet: startup and bandwidth
  - Standard Ethernet: bus based (CSMA/CD)
  - Switched Ethernet: switching does not reduce startup.

- Parallel Programming Model

  - Shared Memory
  - Message Passing

- Parallel Processing Model

  - SPMD
  - Other/s

4.2.1   Computing Nodes Performance

Homogeneous nodes $\Longrightarrow$ no new problems.

Heterogeneous nodes $\Longrightarrow$ two added tasks:

- Specific performance measures

- Processing workload balance

Specific Performance Measures
   1) Specific programs: the same kind of processing but scaled to one processor/node.
This does not mean the parallel program with only one task.
   2) General benchmarks: generalization $\Longrightarrow$?

Linear algebra: given P computers, $comp_i$, $0 \leq i \leq P - 1$, $Mflop/s(comp_i)$, relative computing power, $rp(comp_i)$,

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

Normalized computing power, $np(comp_i)$

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

where

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

Processing Workload Balance

There are many ways: "automatic", dynamic and static processing workload balance. Master-slave could be considered automatic processing workload balance. Dynamic processing workload balance is relatively justified on "unknown" data-dependent systems. Static processing workload balance is strongly "suggested" for linear algebra operations and methods.

Note the relationship among: number of floating points operations and np_i as defined above. Some questions:

- And block processing? (remember flop count is from sequential and *mathematically* defined operation or method).

- What about iterative methods (used on general sparse linear systems)? These methods end when computed solution is *close enough* to the real solution (*convergence*).

4.2.2 Interconnection Network Performance

Standard parameters/indexes: Latency (or startup) and Bandwidth (or data rate). Modeling message time:
$$t(n) = \alpha + \beta n$$
where n is the number of data items, α is the latency, and β is $bndw^{-1}$ where $bndw$ is the data bandwidth. Usually, latency and bandwidth are found experimentally.

If latency is not taken into account, β is the total cost (in time) per data item (and for short messages α is distributed on the data items), i.e.

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

which is almost true when n is large enough ($\alpha \ll \beta n$).

Latency

According to literature, on 100 Mb/s Ethernet the measured latency is about 0,5 ms. On computers with 1 Gflop/s (10^9 floating point operations per second) this means

$$flxlat = 10^9 \times 5 \times 10^{-4} = 5 \times 10^5 = 500000$$

i.e. waiting for *any* message completion implies a time equivalent to 500000 floating point operations. The *main* problem: latency is almost constant and computer performance is enhanced each year (Moore's Law).

Bandwidth

Bandwidth is given in number of data items per second, e.g. 100 Mb/s (100×10^6 bits per second). The time taken to transmit a single data item is easily calculated as β. Even when β is assumed to be the *cost per data item*, for small enough messages (n), the cost per data item is, in fact,

$$\frac{\alpha}{n} + \beta$$

because

$$t(n) = n\left(\frac{\alpha}{n} + \beta\right)$$

and

$$\lim_{n \to \infty} \left(\frac{\alpha}{n} + \beta\right) = \beta$$

Sometimes it is useful to know the message length for which half the data bandwidth is obtained ($n_{1/2}$).

Latency-Bandwidth Law

Given an interconnection network, adding hardware (NIC: Network Interface Card) implies multiplying bandwidth, but latency is constant (in the best case) or worse (drivers-routers complexity).

Ethernet Hardware-Cabling

Performance in bus based cabling (old coaxial cable and hubs) is strongly influenced by the CSMA/CD (Carrier Sense-Multiple Access/Collision Detect) protocol. Simultaneous communications are carried out sequentially: random order and penalized performance.

Switched Ethernet: *just* adding Ethernet switches. Switched Ethernet works *most of the time* as a standard dynamic interconnection network. Some issues:

- Performance: multiple point-to-point operations without restrictions: pairs of computers, performance, and time (collisions are not avoided and are solved as in the standard Ethernet).

- Cost (by 2003-2004). 8 computers: as a hub. 9 to 24 computers: as three hubs. More than 24: depends... Hint: switching cost grows more than linear.

- Cascades: difficult to model performance.

5 LAPACK

From LAPACK Homepage http://www.netlib.org/lapack/

"LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision."

"The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors."

"LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operations kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the L3 BLAS"

From LINPACK Homepage http://www.netlib.org/linpack/
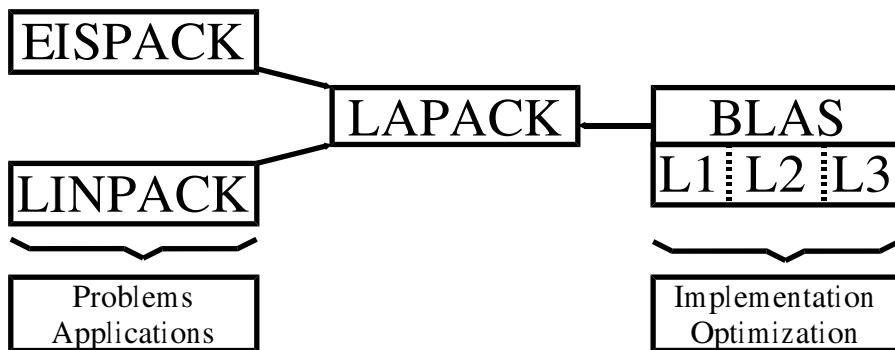
"**LINPACK** is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems. The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems."

From EISPACK Homepage http://www.netlib.org/eispack/

"**EISPACK** is a collection of Fortran subroutines that compute the eigenvalues and eigenvectors of nine classes of matrices: complex general, complex Hermitian, real general, real symmetric, real symmetric banded, real symmetric tridiagonal, special real tridiagonal, generalized real, and generalized real symmetric matices. In addition, two routines are included that use singular value decomposition to solve certain least-squares problems."

Finally, there are a lot of functions (well, subroutines...), each one with a lot of parameters. Classifications (from LAPACK documentation):

- driver routines, each of which solves a complete problem, for example solving a system of linear equations, or computing the eigenvalues of a real symmetric matrix. Users are recommended to use a driver routine if there is one that meets their requirements.

- computational routines, each of which performs a distinct computational task, for example an LU factorization, or the reduction of a real symmetric matrix to tridiagonal form. Each driver routine calls a sequence of computational routines. Users (especially software developers) may need to call computational routines directly to perform tasks, or sequences of tasks, that cannot conveniently be performed by the driver routines.

- Auxiliary routines, which in turn can be classified as follows:

  - routines that perform subtasks of block algorithms – in particular, routines that implement unblocked versions of the algorithms;

  - routines that perform some commonly required low-level computations, for example scaling a matrix, computing a matrix-norm, or generating an elementary Householder matrix; some of these may be of interest to numerical analysts or software developers and could be considered for future additions to the BLAS;

  - a few extensions to the BLAS, such as routines for applying complex plane rotations, or matrix-vector operations involving complex symmetric matrices (the BLAS themselves are not part of LAPACK).

i.e. (approx.)
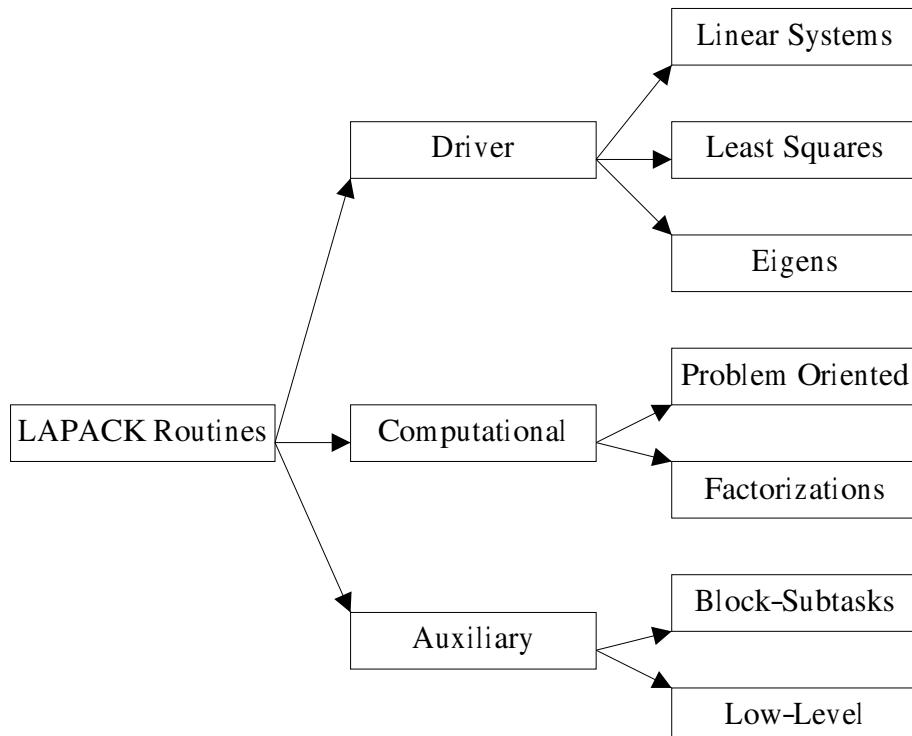
What are the "important" routines? Well, it depends...

- "...the scientists' goals are to solve the challenging problems..." $\Rightarrow$ Driver.

- Performance $\Rightarrow$ Computational. Our work! (?).

Summarizing LAPACK sources and documentation

```
                    ┌─────────────────────────────────┐
                    │ LAPACK Documentation – Sources  │
                    └─────────────────────────────────┘
                           ↙            ↘
      ┌──────────────────────────┐   ┌──────────────────────────┐
      │ Homepage with software   │   │ LAPACK User's Guide      │
      └──────────────────────────┘   └──────────────────────────┘
                                         ↙            ↘
                        ┌──────────────────────┐   ┌──────────────────────┐
                        │ Hardcopy Book        │   │ HTML Book            │
                        └──────────────────────┘   └──────────────────────┘
```
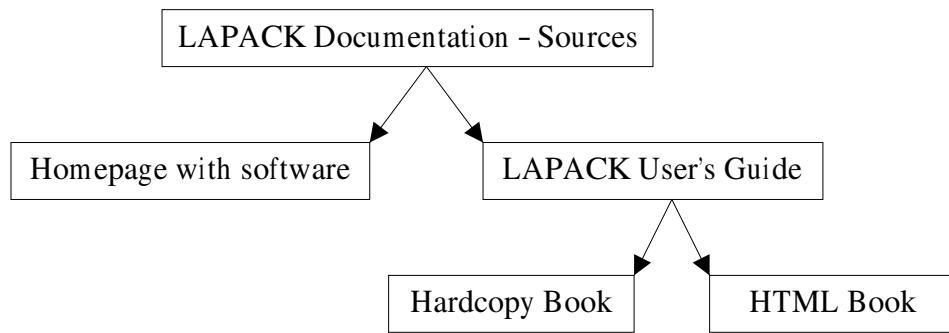
And "lawns" (LAPACK Working Note/s) and A LOT of papers.

6   BLAS

From LAPACK Homepage http://www.netlib.org/lapack/

"LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operations kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the L3 BLAS – a set of specifications for FORTRAN subprograms that do various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides"

Finally, almost everything is written in terms of the BLAS and most of the whole performance depends on the L3 BLAS performance.

From BLAS Homepage http://www.netlib.org/blas/

"This material is based upon work supported by the National Science Foundation under Grant No. ASC-9313958 and DOE Grant No. DE-FG03-94ER25219. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF) or the Department of Energy (DOE)."

So??? ok, information is on its FAQ:

    1.1) What are the BLAS?
    1.2) Publications/references for the BLAS?
    1.3) Is there a Quick Reference Guide to the BLAS available?
    1.4) Are optimized BLAS libraries available?
    1.5) What is ATLAS?
    1.6) Where can I find vendor supplied BLAS?
    1.7) Where can I find the Intel BLAS for Linux?
    1.8) Where can I find Java BLAS?
    1.9) Is there a C interface to the BLAS?
    1.10) Are prebuilt Fortran77 ref implementation BLAS libraries available from Netlib?

What kind of operations are included? Every operation is included in one "Level", as described in the next section.

6.1   BLAS Levels

Subroutines included in BLAS are classified according to its requirements of memory and floating point operations. Assuming

- α and β are scalars.

- x and y are n-elements vectors.

- A, B, and C are square matrices with $n \times n$ elements.

The BLAS are divided in three *levels*:

- Level 1 or L1 BLAS: routines making operations among vectors, thus with access to $O(n)$ data and number of floating point operations also $O(n)$, such as $y = \alpha x + y$.

- Level 2 or L2 BLAS: routines making operations among vectors and matrices, thus with access to $O(n^2)$ data and number of floating point operations also $O(n^2)$, such as $y = \alpha A x + \beta y$.

- Level 3 or L3 BLAS: routines making operations among matrices, thus with access to $O(n^2)$ data and number of floating point operations $O(n^3)$, such as $C = \alpha AB + \beta C$.

Summaryzing:
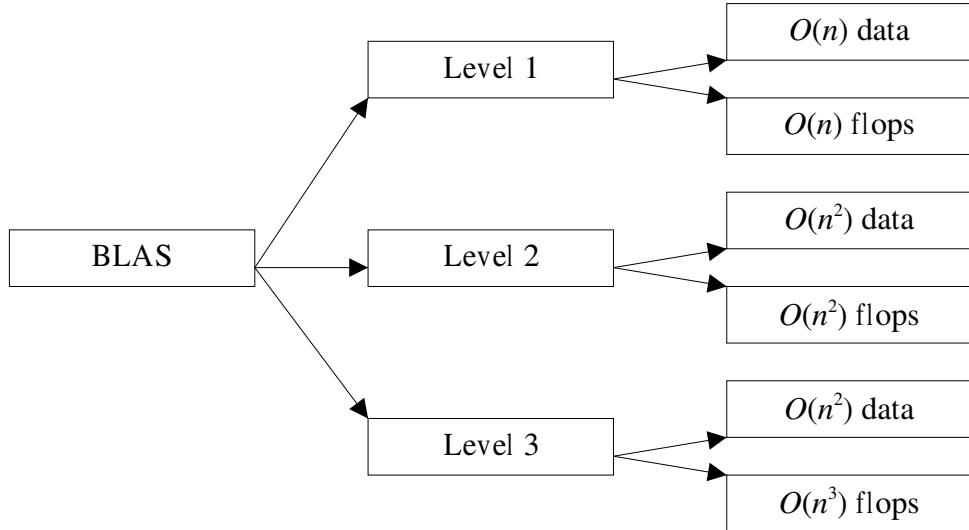
where flops: "number of floating point operations". Note that

- L1 and L2: 1 flop per access.

- L3: n flops per access. !

And this is why

"Finally, almost everything is written in terms of the BLAS and most of the whole performance depends on the L3 BLAS performance."

6.2   Level 3 BLAS

Given that "...most of the whole performance depends on the L3 BLAS performance" it is worth analyzing L3 BLAS in deeper detail. The subroutines defined in this level are:

1) "General" Matrix Multiplication, or matrix multiplication with "general" matrices (_GEMM)

$$C \leftarrow \alpha \; Op(A) \; Op(B) \; + \; \beta \; C$$

where A, B, and C are matrices, α and β are scalars, and $Op(X)$ may be X, X^T or X^H.

2) Matrix multiplication involving a symmetric or Hermitian matrix (_SYMM or _HEMM)

$$C \leftarrow \alpha AB + \beta C \qquad or \qquad C \leftarrow \alpha BA + \beta C$$

where matrix A is symmetric (_SYMM) or Hermitian (_HEMM) and is multiplied at left or right of matrix B depending upon a parameter.

3) Matrix multiplication involving a triangular matrix (_TRMM)

$$B \leftarrow \alpha Op(A)B \qquad or \qquad B \leftarrow \alpha B Op(A)$$

where matrix A is triangular, $Op(X)$ may be X, X^T or X^H, and $Op(A)$ is multiplied at left or right of matrix B depending upon a parameter.

4) Rank-k update of a symmetric or Hermitian matrix (_SYRK or _HERK)

$$C \leftarrow \alpha A Op(A) + \beta C \qquad or \qquad C \leftarrow \alpha Op(A)A + \beta C$$

where matrix A is symmetric (_SYRK) or Hermitian (_HERK), if A is symmetric then $Op(A) = A^T$ and $Op(A) = A^H$ otherwise, and A is multiplied at left or right of matrix B depending upon a parameter.

5) Rank-2k update of a symmetric or Hermitian matrix (_SYR2K or _HER2K)

$$C \leftarrow \alpha A Op(B) + \overline{\alpha} B Op(A) + \beta C \qquad or \qquad C \leftarrow \alpha Op(A)B + \overline{\alpha} Op(B)A + \beta C$$

where
    if matrix C is symmetric (_SYR2K) then $Op(X) = X^T$ and $\alpha \in I\!R$, thus $\overline{\alpha} = \alpha$,
    if matrix C is Hermitian (_HER2K) then $Op(X) = X^H$,
and A is multiplied at left or right of matrix B depending upon a parameter.

6) Solution of triangular systems of equations with multiple right-hand sides (_TRSM)

$$B \leftarrow \alpha Op(A)B \qquad or \qquad B \leftarrow \alpha B Op(A)$$

where matrix A is lower or upper triangular (eventually with unit diagonal), $Op(A)$ may be A^{-1}, A^{-T} or A^{-H}, and $Op(A)$ is multiplied at left or right of matrix B depending upon a parameter.

6.3 Wait, Wait, What about Matrix Multiplication?

Good question, because
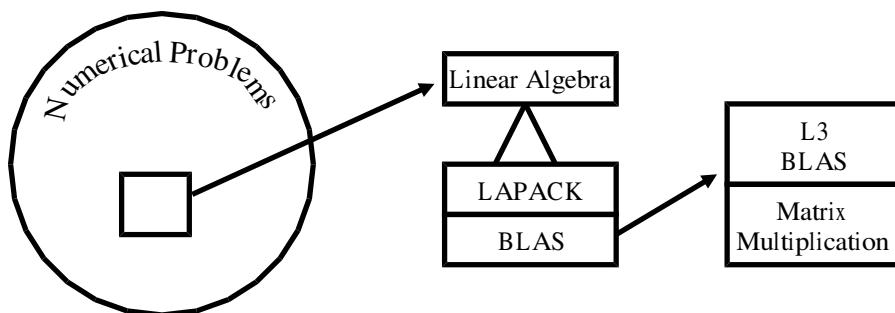
Well, from _GEMM to _XXR2K -items 1) to 5) in the previous enumeration of L3 BLAS description- subroutines are basically matrix multiplications. How are related solutions of triangular systems of equations to matrix multiplications?

    Kågström B., P. Ling, C. Van Loan, "Portable High-Performance GEMM-based Level 3 BLAS", R. F. Sincovec et al., Editor, Parallel Processing for Scientific Computing, Philadelphia, 1993, SIAM, pp. 339- 346.
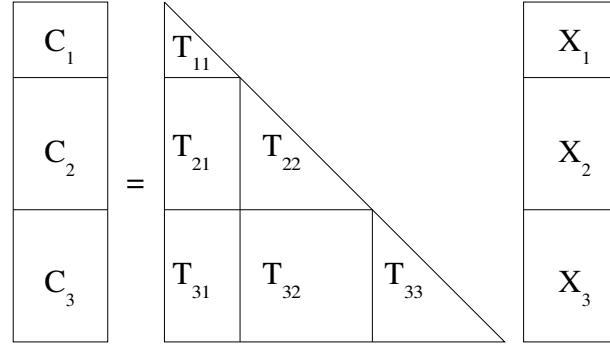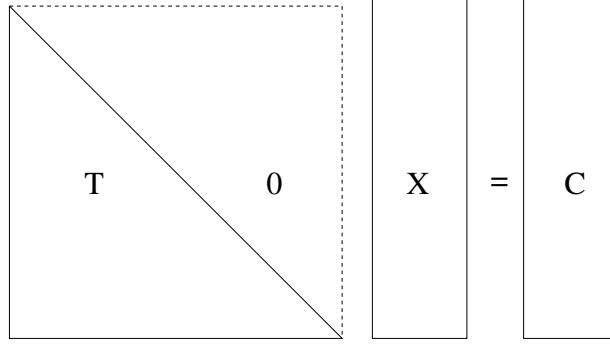
Thus,

$$C_1 = T_{11}X_1 \tag{1}$$
$$C_2 = T_{21}X_1 + T_{22}X_2 \tag{2}$$
$$C_3 = T_{31}X_1 + T_{32}X_2 + T_{33}X_3 \tag{3}$$

or

$$T_{11}X_1 = C_1$$
$$T_{21}X_1 + T_{22}X_2 = C_2$$
$$T_{31}X_1 + T_{32}X_2 + T_{33}X_3 = C_3$$

And? Solving Eq.(1)

$$X_1 = T_{11}^{-1}C_1 \tag{4}$$

It is possible to use X_1 in Eq.(2)

$$C_2 - T_{21}X_1 = T_{22}X_2 \tag{5}$$

because $C_2 - T_{21}X_1$ is known given X_1. Now, solving Eq.(5),

$$X_2 = T_{22}^{-1}(C_2 - T_{21}X_1) \tag{6}$$

it is possible to use X_1 and X_2 in Eq.(3)

$$C_3 - T_{31}X_1 - T_{32}X_2 = T_{33}X_3 \tag{7}$$

which is solved easily as

$$X_3 = T_{33}^{-1}(C_3 - T_{31}X_1 - T_{32}X_2) \tag{8}$$

24

Otherwise, once X_1 is obtained as shown in Eq.(4), it is possible to define

$$
\begin{aligned}
C_2 - T_{21}X_1 &= T_{22}X_2 & (9) \\
C_3 - T_{31}X_1 &= T_{32}X_2 + T_{33}X_3 & (10)
\end{aligned}
$$

which is solved using the same *procedure* and, also, has two *simultaneous* matrix multiplications, i.e.

$$
\begin{aligned}
C_2 &\leftarrow C_2 - T_{21}X_1 \\
C_3 &\leftarrow C_3 - T_{31}X_1
\end{aligned}
$$

Ups! It's just like the _GEMM already defined...

$$
C \leftarrow \alpha\ Op(A)\ Op(B)\ +\ \beta\ C
$$

Do you see the relationship?...

Note: expressions like $T_{11}^{-1}C_1$ do not imply finding out an inverse matrix explicitly, but solving a system of equations with multiple right-hand sides (only unknown values are necessary, not the inverse matrix).

6.4   Matrix Multiplication, L3 BLAS and Performance

Given that "...most of the whole performance depends on the L3 BLAS performance" and everything in L3 BLAS may be GEMM-based, the BIG question is:

Matrix multiplication performance is the whole performance to be obtained by the applications?

By the way: What about parallel performance? And parallel performance on clusters?

7   Examples from LAPACK and (L3) BLAS

From LAPACK man pages:

```
NAME
      SGETRF  -  compute an LU factorization of a general M-by-N
      matrix A using partial pivoting with row interchanges

SYNOPSIS
      SUBROUTINE SGETRF( M, N, A, LDA, IPIV, INFO )

          INTEGER         INFO, LDA, M, N

          INTEGER         IPIV( * )

          REAL            A( LDA, * )

PURPOSE
      SGETRF computes an LU factorization of  a  general  M-by-N
      matrix  A  using  partial  pivoting with row interchanges.
      The factorization has the form
         A = P * L * U
      where P is a permutation matrix,  L  is  lower  triangular
      with  unit diagonal elements (lower trapezoidal if m > n),
      and U is upper triangular (upper  trapezoidal  if  m < n).

      This is the right-looking Level  3  BLAS  version  of  the
      algorithm.


ARGUMENTS
      M         (input) INTEGER
                The number of rows of the matrix A.  M >= 0.

      N         (input) INTEGER
                The number of columns of the matrix A.  N >= 0.

      A         (input/output) REAL array, dimension (LDA,N)
                On  entry,  the  M-by-N matrix to be factored.  On
                exit, the factors L and U from the factorization A
                =  P*L*U;  the unit diagonal elements of L are not
                stored.

      LDA       (input) INTEGER
                The leading dimension of  the  array  A.   LDA  >=
```

```
                 max(1,M).

     IPIV    (output) INTEGER array, dimension (min(M,N))
             The  pivot  indices; for 1 <= i <= min(M,N), row i
             of the matrix was interchanged with row IPIV(i).

     INFO    (output) INTEGER
             = 0:  successful exit
             < 0:  if INFO = -i, the i-th argument had an ille
             gal value
             >  0:   if  INFO  = i, U(i,i) is exactly zero. The
             factorization has been completed, but the factor U
             is  exactly  singular,  and  division by zero will
             occur if it is used to solve  a  system  of  equa
             tions.
```

Two details: LDA and REAL A(LDA, *). Well... the third detail: name? From LAPACK online documentation:

Naming Scheme
The name of each LAPACK routine is a coded specification of its function (within the very tight limits of standard Fortran 77 6-character names).

All driver and computational routines have names of the form XYYZZZ, where for some driver routines the 6th character is blank.

The first letter, X, indicates the data type as follows:
    S REAL
    D DOUBLE PRECISION
    C COMPLEX
    Z COMPLEX*16 or DOUBLE COMPLEX

When we wish to refer to a LAPACK routine generically, regardless of data type, we replace the first letter by "x". Thus xGESV refers to any or all of the routines SGESV, CGESV, DGESV and ZGESV.

The next two letters, YY, indicate the type of matrix (or of the most significant matrix). Most of these two-letter codes apply to both real and complex matrices; a few apply specifically to one or the other, as indicated in Table 2.1.

Table 2.1: Matrix types in the LAPACK naming scheme
    BD bidiagonal
    DI diagonal
    GB general band
    GE general (i.e., unsymmetric, in some cases rectangular)
    GG general matrices, generalized problem (i.e., a pair of general matrices)
    GT general tridiagonal
    HB (complex) Hermitian band
    HE (complex) Hermitian

HG upper Hessenberg matrix, generalized problem (i.e a Hessenberg and a triangular matrix)

HP (complex) Hermitian, packed storage

HS upper Hessenberg

OP (real) orthogonal, packed storage

OR (real) orthogonal

PB symmetric or Hermitian positive definite band

PO symmetric or Hermitian positive definite

PP symmetric or Hermitian positive definite, packed storage

PT symmetric or Hermitian positive definite tridiagonal

SB (real) symmetric band

SP symmetric, packed storage

ST (real) symmetric tridiagonal

SY symmetric

TB triangular band

TG triangular matrices, generalized problem (i.e., a pair of triangular matrices)

TP triangular, packed storage

TR triangular (or in some cases quasi-triangular)

TZ trapezoidal

UN (complex) unitary

UP (complex) unitary, packed storage

When we wish to refer to a class of routines that performs the same function on different types of matrices, we replace the first three letters by "xyy". Thus xyySVX refers to all the expert driver routines for systems of linear equations that are listed in Table 2.2.

The last three letters ZZZ indicate the computation performed. Their meanings will be explained in Section 2.4. For example, SGEBRD is a single precision routine that performs a bidiagonal reduction (BRD) of a real general matrix.

The names of auxiliary routines follow a similar scheme except that the 2nd and 3rd characters YY are usually LA (for example, SLASCL or CLARFG). There are two kinds of exception. Auxiliary routines that implement an unblocked version of a block algorithm have similar names to the routines that perform the block algorithm, with the sixth character being "2" (for example, SGETF2 is the unblocked version of SGETRF). A few routines that may be regarded as extensions to the BLAS are named according to the BLAS naming schemes (for example, CROT, CSYR).

And, finally, on "Section 2.4"

xyyTRF: factorize (obviously not needed for triangular matrices);

From BLAS man pages:

NAME
       SGEMM - perform one of the matrix-matrix operations   C :=
       alpha*op(A)*op(B) + beta*C,

SYNOPSIS
       SUBROUTINE SGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA,
                          B, LDB, BETA, C, LDC)

          CHARACTER*1 TRANSA, TRANSB

          INTEGER   M, N, K, LDA, LDB, LDC

          REAL   ALPHA, BETA

          REAL   A(LDA, *), B(LDB, *), C(LDC, *)

PURPOSE
       SGEMM performs one of the matrix-matrix operations

       where op(X) is one of

          op(X) = X   or   op(X) = X',

       alpha and beta are scalars, and A, B and C are matrices,
       with op(A) an m by k matrix, op(B) a k by n matrix
       and C an m by n matrix.

PARAMETERS
       TRANSA - CHARACTER*1. On entry, TRANSA specifies the form
       of op(A) to be used in the matrix multiplication as fol
       lows:

       TRANSA = 'N' or 'n', op(A) = A.

       TRANSA = 'T' or 't', op(A) = A'.

       TRANSA = 'C' or 'c', op(A) = A'.

       Unchanged on exit.

       TRANSB - CHARACTER*1. On entry, TRANSB specifies the form
       of op(B) to be used in the matrix multiplication as fol

lows:

TRANSB = 'N' or 'n', op(B) = B.

TRANSB = 'T' or 't', op(B) = B'.

TRANSB = 'C' or 'c', op(B) = B'.

Unchanged on exit.

M   - INTEGER.
        On entry, M   specifies the number of rows of
        the matrix op(A) and of the   matrix   C.   M
        must be at least zero. Unchanged on exit.

N   - INTEGER.
        On entry, N specifies the number of columns of
        the matrix op(B) and the number of columns of the
        matrix C. N must be at least zero. Unchanged on
        exit.

K   - INTEGER.
        On entry, K specifies the number of columns of
        the matrix op(A) and the number of rows of the
        matrix op(B). K must be at least   zero.
        Unchanged on exit.

ALPHA   - REAL   .
        On   entry,   ALPHA specifies   the scalar alpha.
        Unchanged on exit.

A   - REAL   array of DIMENSION (LDA, ka),
where ka is
        k when TRANSA = 'N' or 'n', and is   m   other
        wise. Before entry with TRANSA = 'N' or 'n', the
        leading m by k part of the array A   must contain
        the matrix A, otherwise the leading k by m part
        of the array   A   must contain   the matrix A.
        Unchanged on exit.

LDA   - INTEGER.
        On entry, LDA specifies the first dimension of A as
        declared in the calling (sub) program. When TRANSA
        = 'N' or 'n' then LDA must be at least max(1, m
        ), otherwise LDA must be at least max(1, k).
        Unchanged on exit.

B   - REAL   array of DIMENSION (LDB, kb),

```
where kb is
        n    when   TRANSB  = 'N' or 'n',  and is  k   other
        wise.  Before entry with  TRANSB = 'N' or 'n',   the
        leading   k by n part of the array  B   must contain
        the matrix  B,  otherwise the leading  n by k  part
        of  the  array  B   must  contain   the matrix B.
        Unchanged on exit.

LDB     - INTEGER.
        On entry, LDB specifies the first dimension of B as
        declared in the calling (sub) program. When  TRANSB
        = 'N' or 'n' then LDB must be at least  max(  1,  k
        ),  otherwise  LDB  must be at least  max( 1, n ).
        Unchanged on exit.

BETA    - REAL              .
        On entry, BETA  specifies the scalar  beta.   When
        BETA  is supplied as zero then C need not be set on
        input.  Unchanged on exit.

C       - REAL               array of DIMENSION ( LDC, n ).
        Before entry, the leading  m  by  n   part  of  the
        array  C  must contain the matrix  C,   except when
        beta  is zero, in which case C need not be  set  on
        entry.  On  exit,  the array  C  is overwritten by
        the  m by n  matrix ( alpha*op( A  )*op( B ) +
        beta*C ).

LDC     - INTEGER.
        On entry, LDC specifies the first dimension of C as
        declared in  the  calling  (sub)  program.   LDC
        must   be   at   least max( 1, m ).  Unchanged on
        exit.

        Level 3 Blas routine.

        -- Written  on  8-February-1989.   Jack  Dongarra,
        Argonne  National Laboratory.  Iain Duff, AERE Har
        well.  Jeremy Du Croz, Numerical  Algorithms  Group
        Ltd.   Sven  Hammarling, Numerical Algorithms Group
        Ltd.
```

8   A Step Forward: ScaLAPACK

From ScaLAPACK homepage http://www.netlib.org/scalapack

"The ScaLAPACK project was a collaborative effort involving several institutions:

- Oak Ridge National Laboratory

- Rice University

- University of California, Berkeley

- University of California, Los Angeles

- University of Illinois

- University of Tennessee, Knoxville

and comprised four components:

- dense and band matrix software (ScaLAPACK)

- large sparse eigenvalue software (PARPACK and ARPACK)

- sparse direct systems software (CAPSS and MFACT)

- preconditioners for large sparse iterative solvers (ParPre)

Funding for this effort came in part from DARPA, DOE, NSF, and CRPC."

From ScaLAPACK (again) homepage http://www.netlib.org/scalapack/scalapack_home (ok, it is some confusing)

"The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory MIMD parallel computers. It is currently written in a Single-Program-Multiple-Data style using explicit message passing for interprocessor communication. It assumes matrices are laid out in a two-dimensional block cyclic decomposition."

This first paragraph has a lot of information:

- Subset of LAPACK routines.

- Distributed memory MIMD parallel computers.

- SPMD.

- Explicit message passing.

- Matrices data distribution!

Most of the documentation (and much of the code) is like LAPACK. In fact, the "graphical view" of the ScaLAPACK sources and documentation is
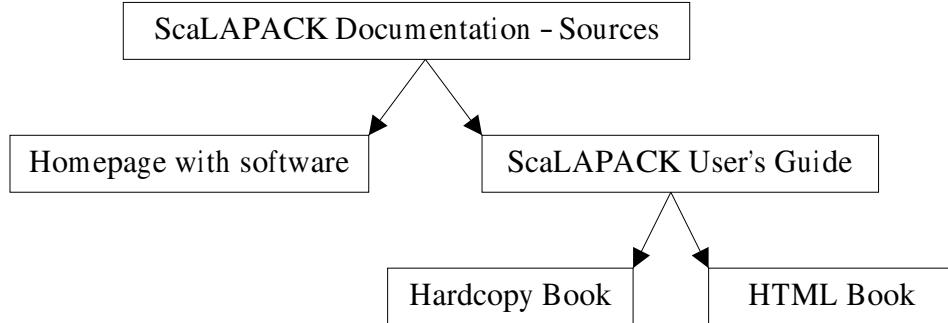
And some "lawns" (LAPACK Working Note/s) and A LOT of papers.

The *official* relationship among ScaLAPACK, LAPACK and BLAS is shown in the ScaLA-PACK homepage, which can be summarized as
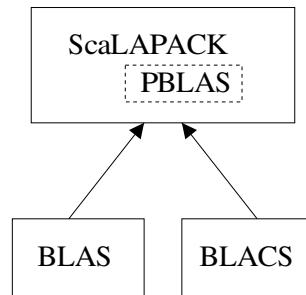
Added by ScaLAPACK

- PBLAS.

- BLACS.

A little introduction to "...two-dimensional block cyclic decomposition..." Having a matrix and a 2-Dimensional array of processors:

The resulting matrix distribution is

A

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}
a_{60}	a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}
a_{70}	a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{67}

Processors

	0	**1**
0	$a_{00}\ a_{02}\ a_{04}\ a_{06}$ $a_{30}\ a_{32}\ a_{34}\ a_{36}$ $a_{60}\ a_{62}\ a_{64}\ a_{66}$	$a_{01}\ a_{03}\ a_{05}\ a_{07}$ $a_{31}\ a_{33}\ a_{35}\ a_{37}$ $a_{61}\ a_{63}\ a_{65}\ a_{67}$
1	$a_{10}\ a_{12}\ a_{14}\ a_{16}$ $a_{40}\ a_{42}\ a_{44}\ a_{46}$ $a_{40}\ a_{72}\ a_{74}\ a_{76}$	$a_{11}\ a_{13}\ a_{15}\ a_{17}$ $a_{41}\ a_{43}\ a_{45}\ a_{47}$ $a_{71}\ a_{73}\ a_{75}\ a_{77}$
2	$a_{20}\ a_{22}\ a_{24}\ a_{26}$ $a_{50}\ a_{52}\ a_{54}\ a_{56}$	$a_{21}\ a_{23}\ a_{25}\ a_{27}$ $a_{51}\ a_{53}\ a_{55}\ a_{57}$

Reason: matrix factorization algorithms. Why does matrix distribution becomes visible to the user?

What about an example?

PSGETRF(1)   )   PSGETRF(1)

NAME
        PSGETRF - compute an LU factorization of a general M-by-N
        distributed matrix sub(A) = (IA:IA+M-1,JA:JA+N-1) using
        partial pivoting with row interchanges

SYNOPSIS
        SUBROUTINE PSGETRF(M, N, A, IA, JA, DESCA, IPIV, INFO)

            INTEGER   IA, INFO, JA, M, N

            INTEGER   DESCA(*), IPIV(*)

            REAL   A(*)

PURPOSE
        PSGETRF computes an LU factorization of a general M-by-N
        distributed matrix sub(A) = (IA:IA+M-1,JA:JA+N-1) using
        partial pivoting with row interchanges. The factorization
        has the form sub(A) = P * L * U, where P is a permuta
        tion matrix, L is lower triangular with unit diagonal ele-
        ments (lower trapezoidal if m > n), and U is upper trian
        gular (upper trapezoidal if m < n). L and U are stored in
        sub(A).

        This is the right-looking Parallel Level 3 BLAS version of
        the algorithm.

```
Notes
=====


Each  global  data  object  is  described by an associated
description vector.  This vector  stores  the  information
required  to  establish the mapping between an object ele
ment and its corresponding process and memory location.

Let A be a generic term for any  2D  block  cyclicly  dis
tributed  array.   Such  a  global array has an associated
description vector DESCA.  In the following comments,  the
character _ should be read as "of the global array".


NOTATION          STORED IN       EXPLANATION
---------------                          --------------
---------------------------------------       DTYPE_A(global)
DESCA( DTYPE_ )The descriptor type.  In this case,
                                DTYPE_A = 1.
CTXT_A  (global) DESCA( CTXT_ ) The BLACS context handle,
indicating

                                the BLACS process grid A is
distribu-

                                ted   over.   The   context
itself is glo-

                                bal, but  the  handle  (the
integer

                                value) may vary.
M_A     (global) DESCA( M_ )    The number of rows in the
global

                                array A.
N_A    (global) DESCA( N_ )    The number  of  columns  in
the global

                                array A.
MB_A   (global) DESCA( MB_ )   The blocking factor used to
distribute

                                the rows of the array.
NB_A   (global) DESCA( NB_ )   The blocking factor used to
distribute

                                the columns of the array.
RSRC_A  (global) DESCA( RSRC_ ) The process row over which
the first

                                row of the array A is  dis
tributed.  CSRC_A (global) DESCA( CSRC_ ) The process col
umn over which the

                                first column of the array A
is

                                distributed.
LLD_A   (local)  DESCA( LLD_ ) The leading dimension of
```

the local

array.   LLD_A   >=
MAX(1,LOCr(M_A)).

Let K be the number of rows or columns of a distributed
matrix, and assume that its process grid has dimension p x
q.
LOCr(K) denotes the number of elements of K that a pro
cess would receive if K were distributed over the p pro
cesses of its process column.
Similarly, LOCc(K) denotes the number of elements of K
that a process would receive if K were distributed over
the q processes of its process row.
The values of LOCr() and LOCc() may be determined via a
call to the ScaLAPACK tool function, NUMROC:
        LOCr(M) = NUMROC(M, MB_A, MYROW, RSRC_A, NPROW
),
        LOCc(N) = NUMROC(N, NB_A, MYCOL, CSRC_A, NPCOL
). An upper bound for these quantities may be computed
by:
        LOCr(M) <= ceil(ceil(M/MB_A)/NPROW)*MB_A
        LOCc(N) <= ceil(ceil(N/NB_A)/NPCOL)*NB_A

This routine requires square block decomposition (MB_A =
NB_A).

ARGUMENTS
        M   (global input) INTEGER
                The number of rows to be operated on, i.e. the
                number of rows of the distributed submatrix sub(A
). M >= 0.

        N   (global input) INTEGER
                The number of columns to be operated on, i.e. the
                number of columns of the distributed submatrix
                sub(A). N >= 0.

        A   (local input/local output) REAL pointer into the
                local memory to an array of dimension (LLD_A,
                LOCc(JA+N-1)).   On entry, this array contains the
                local pieces of the M-by-N distributed matrix sub(
                A) to be factored. On exit, this array contains
                the local pieces of the factors L and U from the
                factorization sub(A) = P*L*U; the unit diagonal
                ele- ments of L are not stored.

        IA   (global input) INTEGER

36

The row index in the global array A indicating the
first row of sub(A).

JA   (global input) INTEGER
          The column index in the global array A indicating
          the first column of sub(A).

DESCA   (global and local input) INTEGER array of dimen
sion DLEN_.
          The array descriptor for the distributed matrix A.

IPIV   (local   output) INTEGER array, dimension (
LOCr(M_A)+MB_A)
          This array contains the pivoting information.
          IPIV(i) -> The global row local row i was swapped
          with.   This array is tied to the distributed
          matrix A.

INFO   (global output) INTEGER
          = 0: successful exit
          < 0: If the i-th argument is an array and the j-
          entry had an illegal value, then INFO =
          -(i*100+j), if the i-th argument is a scalar and
          had an illegal value, then INFO = -i. > 0: If
          INFO = K, U(IA+K-1,JA+K-1) is exactly zero.   The
          factorization has been completed, but the factor U
          is exactly singular, and division by zero will
          occur if it is used to solve a system of equa
          tions.

9 Specific Algorithms for Simple Operations

First: there are not complex operations. However, the simplest (and most important?): matrix multiplication. Given

$$A \in I\!R^{m \times k}$$

and

$$B \in I\!R^{k \times n}$$

where the elements of matrix A are denoted as

$$a_{ij}, 1 \leq i \leq m, 1 \leq j \leq k$$

and the elements of matrix B are denoted as

$$b_{ij}, 1 \leq i \leq k, 1 \leq j \leq n$$

matrix

$$C \in I\!R^{m \times n}$$

with elements denoted as

$$c_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$$

from the multiplication

$$C = A \times B$$

is defined by

$$c_{ij} = \sum_{r=1}^{k} a_{ik} b_{kj}$$

If $m = n = k$, the number of floating point operations is $O(n^3)$. Furthermore, the exact number of floating point operations, $flops\,MM$, is

$$flops\,MM = 2n^3 - n^2$$

having square matrices of order n. The general parallel matrix multiplication algorithms:

- For multiprocessors (shared memory parallel computers).

- For multicomputers (distributed memory parallel computers).

- For Clusters!

9.1 Matrix Multiplication on Multiprocessors

The simplest
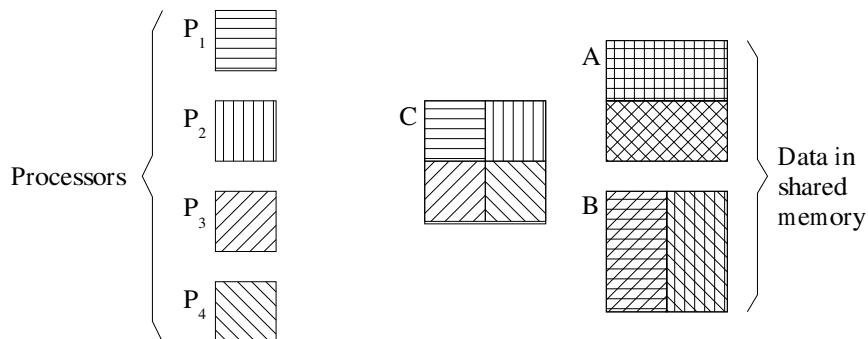
With recursion
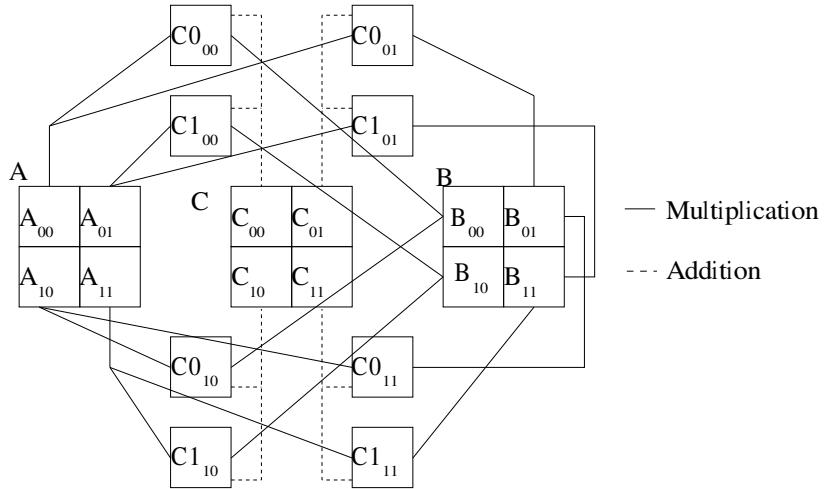

```
mat_mul(A, B, C, s)  /* C = A×B */
/* A, B: operands */
/* C    : result */
/* s    : matrices (square) size */
{
    if (sequential multiplication)
    {
        C = A×B;
    }
    else
    {
        mat_mul(A00, B00, C000, s/2);    /* (1) */
        mat_mul(A01, B10, C100, s/2);    /* (2) */
        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;
}
```

Is $C_{00} = C0_{00} + C1_{00}$?

$$C0_{00} = A_{00} \text{ x } B_{00}$$
$$C1_{00} = A_{01} \text{ x } B_{10}$$

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

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

Thus, $C0_{00} + C1_{00}$

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

Important: this is block processing.

Avoiding extra memory requirements and including data dependence

```
mat_mul_sum(A, B, C, s)  /* C = AxB + C */
/* A, B: operands */
/* C   : result */
/* s   : matrices (square) size */
{
   if (sequential multiplication)
   {
      C = AxB + C;
   }
   else
   {
      mat_mul_sum(A  , B  , C  , s/2);    /* (1) */
                   00   00   00
      mat_mul_sum(A  , B  , C  , s/2);    /* (2) */
                   01   10   00
      mat_mul_sum(A  , B  , C  , s/2);    /* (3) */
                   00   01   01
      mat_mul_sum(A  , B  , C  , s/2);    /* (4) */
                   01   11   01
      mat_mul_sum(A  , B  , C  , s/2);    /* (5) */
                   10   00   10
      mat_mul_sum(A  , B  , C  , s/2);    /* (6) */
                   11   10   10
      mat_mul_sum(A  , B  , C  , s/2);    /* (7) */
                   10   01   11
      mat_mul_sum(A  , B  , C  , s/2);    /* (8) */
                   11   11   11
   }
}
```

Strassen's Method (works? flop count?)

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

A
A_{00}	A_{01}
A_{10}	A_{11}

B
B_{00}	B_{01}
B_{10}	B_{11}

C
C_{00}	C_{01}
C_{10}	C_{11}

a) Computing with submatrices   b) Matrix Partition

9.2 Matrix Multiplication on Multicomputers

On distributed memory parallel computers: basically Cannon & Fox. However,

a) First step

b) Second step

$$c_{00}^1 = a_{00} \times b_{00}$$

$$c_{00}^2 = a_{00} \times b_{00} + a_{01} \times b_{10}$$

$$c_{10}^2 = a_{10} \times b_{00} \qquad c_{01}^2 = a_{00} \times b_{01}$$
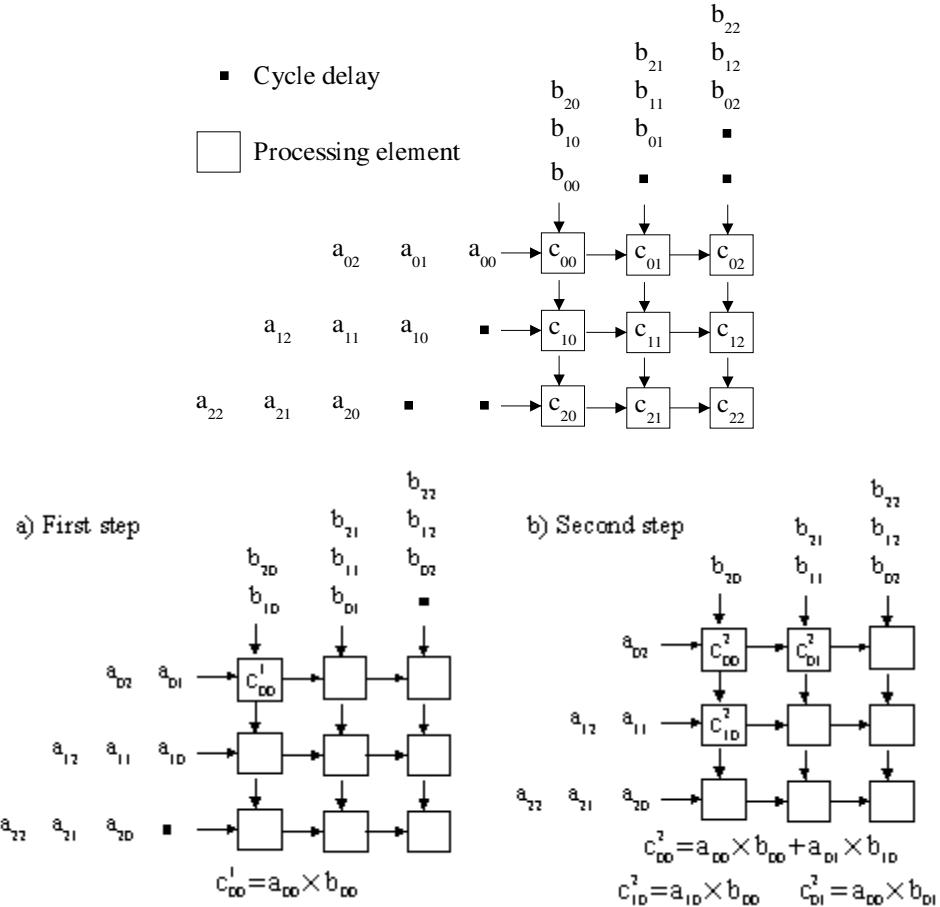
The most common interprocessor network
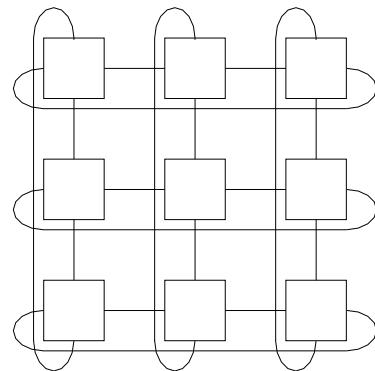
Cannon: relocation and shifts

a_{00} b_{00} c_{00}	a_{01} b_{01} c_{01}	a_{02} b_{02} c_{02}		a_{00} b_{00} c_{00}	a_{01} b_{11} c_{01}	a_{02} b_{22} c_{02}
a_{10} b_{10} c_{10}	a_{11} b_{11} c_{11}	a_{12} b_{12} c_{12}		a_{11} b_{10} c_{10}	a_{12} b_{21} c_{11}	a_{10} b_{02} c_{12}
a_{20} b_{20} c_{20}	a_{21} b_{21} c_{21}	a_{22} b_{22} c_{22}		a_{22} b_{20} c_{20}	a_{20} b_{01} c_{21}	a_{21} b_{12} c_{22}
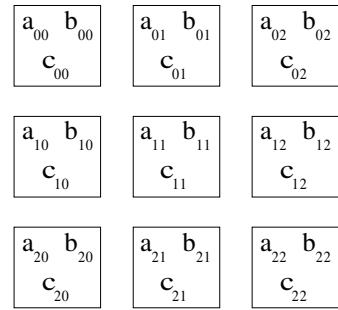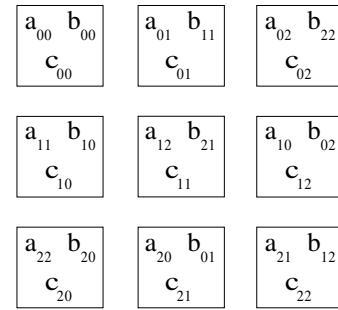
a) Initial Distribution   a) Initial Alignment

Fox: broadcast and shift

$$c_{00}^{(1)} = a_{00} \times b_{00}; \quad c_{01}^{(1)} = a_{00} \times b_{01}; \quad c_{02}^{(1)} = a_{00} \times b_{02}$$

$$c_{10}^{(1)} = a_{11} \times b_{10}; \quad c_{11}^{(1)} = a_{11} \times b_{11}; \quad c_{12}^{(1)} = a_{11} \times b_{12}$$

$$c_{20}^{(1)} = a_{22} \times b_{20}; \quad c_{21}^{(1)} = a_{22} \times b_{21}; \quad c_{22}^{(1)} = a_{22} \times b_{22}$$

a) Broadcast   b) Local computing   a) Shift

$$c_{00}^{(2)} = c_{00}^{(1)} + a_{01} \times b_{10}; \quad c_{01}^{(2)} = c_{01}^{(1)} + a_{01} \times b_{11}; \quad c_{02}^{(2)} = c_{02}^{(1)} + a_{01} \times b_{12}$$

$$c_{10}^{(2)} = c_{10}^{(1)} + a_{12} \times b_{20}; \quad c_{11}^{(2)} = c_{11}^{(1)} + a_{12} \times b_{21}; \quad c_{12}^{(2)} = c_{12}^{(1)} + a_{12} \times b_{22}$$

$$c_{20}^{(2)} = c_{20}^{(1)} + a_{20} \times b_{00}; \quad c_{21}^{(2)} = c_{21}^{(1)} + a_{20} \times b_{01}; \quad c_{22}^{(2)} = c_{22}^{(1)} + a_{20} \times b_{02}$$
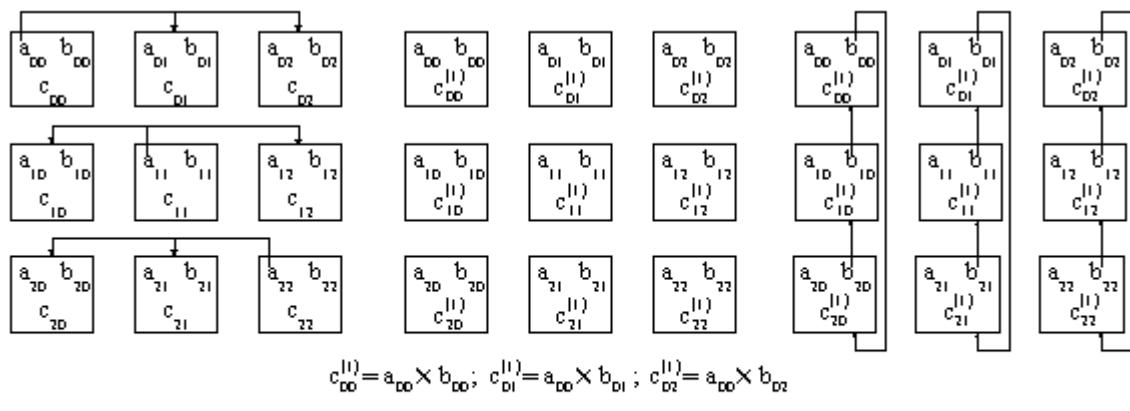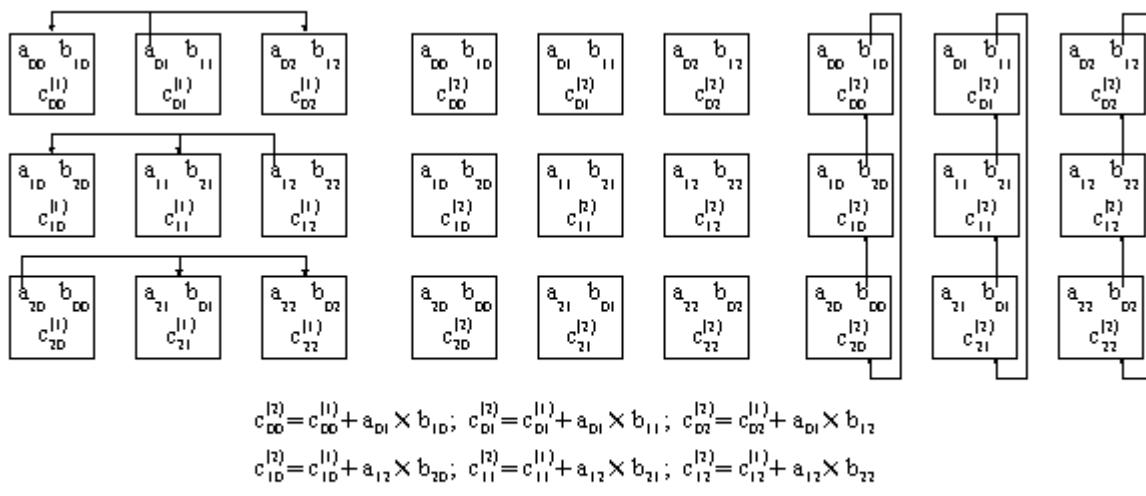
a) Broadcast   b) Local computing   a) Shift

Three dimensions and more: DNS (data replication)

a) Processors identification.   b) Distribution of columns of A.   c) Broadcast by Planes.
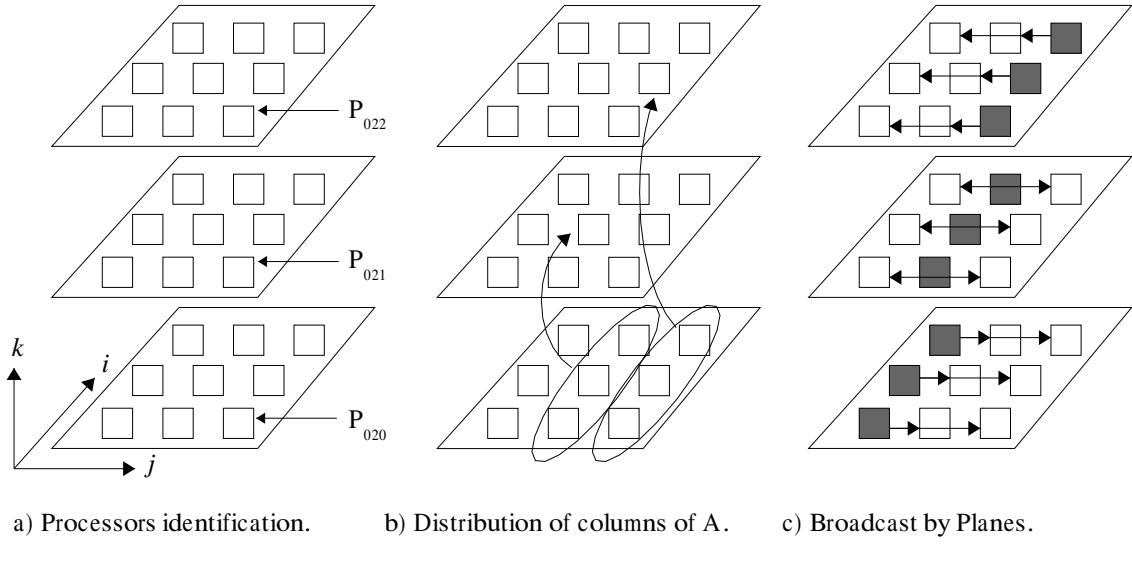
9.3   Matrix Multiplication on Clusters

Some previous work first, which is based on previous algorithms for multicomputers. Later, the specific proposal for (Ethernet-interconnected) clusters.

9.4   Matrix Multiplication Already Proposed for DMPC (and Clusters)

PUMMA-SUMMA-DIMMA (MMA: Matrix Multiplication Algorithm). SUMMA: Scalable Universal MMA, almost directly used in ScaLAPACK. Taking into account "two-dimensional block cyclic decomposition" is losely based on Fox's algorithm and losely resembling Cannon's algorithm data communication pattern. Data distribution:
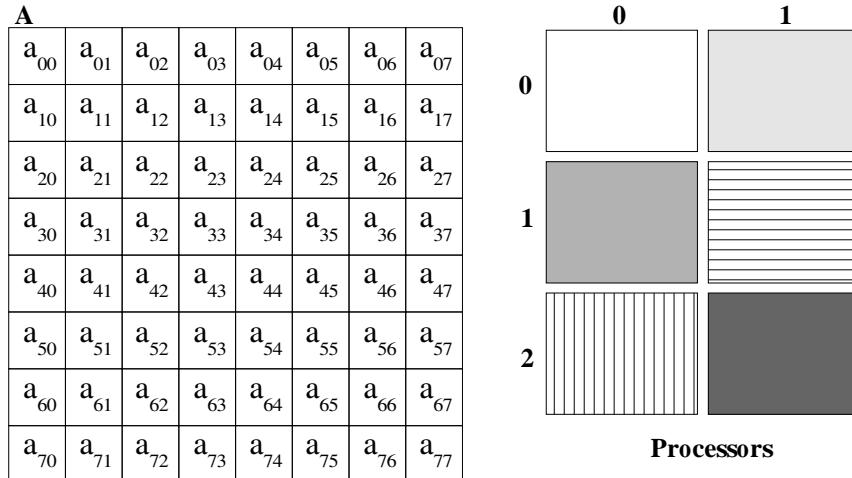
The resulting matrix distribution is
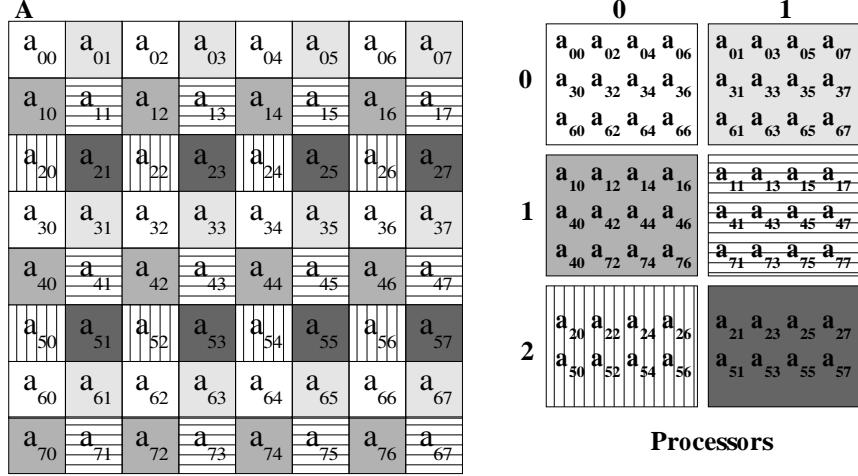
SUMMA in pseudocode (assuming k is the "common" index, columns of A and rows of B) without blocking:

```
for(i = 0; i < k ; i++)
{
    Send k-th column of A in a row broadcast
    Send k-th row of B in a column broadcast
    Multiply k-th column by k-th row
}
```

which uses broadcasts as Fox's algorithm.

The iterations on processor 0:

$\overrightarrow{Broadcasts_0}$

a_{00}	b_{00}	b_{02}	b_{04}	b_{06}
a_{00}	$c_{00}^{(0)}$	$c_{02}^{(0)}$	$c_{04}^{(0)}$	$c_{06}^{(0)}$
a_{30}	$c_{30}^{(0)}$	$c_{32}^{(0)}$	$c_{34}^{(0)}$	$c_{36}^{(0)}$
a_{60}	$c_{60}^{(0)}$	$c_{62}^{(0)}$	$c_{64}^{(0)}$	$c_{66}^{(0)}$

$\overrightarrow{Broadcasts_1}$

a_{01}	b_{10}	b_{12}	b_{14}	b_{16}
a_{01}	$c_{00}^{(1)}$	$c_{02}^{(1)}$	$c_{04}^{(1)}$	$c_{06}^{(1)}$
a_{31}	$c_{30}^{(1)}$	$c_{32}^{(1)}$	$c_{34}^{(1)}$	$c_{36}^{(1)}$
a_{61}	$c_{60}^{(1)}$	$c_{62}^{(1)}$	$c_{64}^{(1)}$	$c_{66}^{(1)}$

$\overrightarrow{Broadcasts_2}$

| a_{02}	b_{20}	b_{22}	b_{24}	b_{26}
a_{02}	$c_{00}^{(2)}$	$c_{02}^{(2)}$	$c_{04}^{(2)}$	$c_{06}^{(2)}$
a_{32}	$c_{30}^{(2)}$	$c_{32}^{(2)}$	$c_{34}^{(2)}$	$c_{36}^{(2)}$
a_{62}	$c_{60}^{(2)}$	$c_{62}^{(2)}$	$c_{64}^{(2)}$	$c_{66}^{(2)}$

$\overrightarrow{Broadcasts_3}$

a_{03}	b_{30}	b_{32}	b_{34}	b_{36}
a_{03}	$c_{00}^{(3)}$	$c_{02}^{(3)}$	$c_{04}^{(3)}$	$c_{06}^{(3)}$
a_{33}	$c_{30}^{(3)}$	$c_{32}^{(3)}$	$c_{34}^{(3)}$	$c_{36}^{(3)}$
a_{63}	$c_{60}^{(3)}$	$c_{62}^{(3)}$	$c_{64}^{(3)}$	$c_{66}^{(3)}$

$\overrightarrow{Broadcasts_4}$

a_{04}	b_{40}	b_{42}	b_{44}	b_{46}
a_{04}	$c_{00}^{(4)}$	$c_{02}^{(4)}$	$c_{04}^{(4)}$	$c_{06}^{(4)}$
a_{34}	$c_{30}^{(4)}$	$c_{32}^{(4)}$	$c_{34}^{(4)}$	$c_{36}^{(4)}$
a_{64}	$c_{60}^{(4)}$	$c_{62}^{(4)}$	$c_{64}^{(4)}$	$c_{66}^{(4)}$

$\overrightarrow{Broadcasts_5}$

a_{05}	b_{50}	b_{52}	b_{54}	b_{56}
a_{05}	$c_{00}^{(5)}$	$c_{02}^{(5)}$	$c_{04}^{(5)}$	$c_{06}^{(5)}$
a_{35}	$c_{30}^{(5)}$	$c_{32}^{(5)}$	$c_{34}^{(5)}$	$c_{36}^{(5)}$
a_{65}	$c_{60}^{(5)}$	$c_{62}^{(5)}$	$c_{64}^{(5)}$	$c_{66}^{(5)}$

$\overrightarrow{Broadcasts_6}$

a_{06}	b_{60}	b_{62}	b_{64}	b_{66}
a_{06}	$c_{00}^{(6)}$	$c_{02}^{(6)}$	$c_{04}^{(6)}$	$c_{06}^{(6)}$
a_{36}	$c_{30}^{(6)}$	$c_{32}^{(6)}$	$c_{34}^{(6)}$	$c_{36}^{(6)}$
a_{66}	$c_{60}^{(6)}$	$c_{62}^{(6)}$	$c_{64}^{(6)}$	$c_{66}^{(6)}$

$\overrightarrow{Broadcasts_7}$

a_{07}	b_{70}	b_{72}	b_{74}	b_{76}
a_{07}	$c_{00}^{(7)}$	$c_{02}^{(7)}$	$c_{04}^{(7)}$	$c_{06}^{(7)}$
a_{37}	$c_{30}^{(7)}$	$c_{32}^{(7)}$	$c_{34}^{(7)}$	$c_{36}^{(7)}$
a_{67}	$c_{60}^{(7)}$	$c_{62}^{(7)}$	$c_{64}^{(7)}$	$c_{66}^{(7)}$

And this is the final result.

(**1**) Focusing on granularity and local processing performance, matrix blocks or submatrices are used instead of single elements: $x_{ij} \to X_{ij}$. It is not very clear how to define block size (combination of local computing performance and granularity).

(2) Focusing on static interconnection networks (classical on traditional multicomputers), broadcasts are not easily implemented (poor performance is expected a *priori*). Broadcasts are *transformed into* (and the whole algorithm) multiple and pipelined point-to-point messages through the ring of columns or rows.
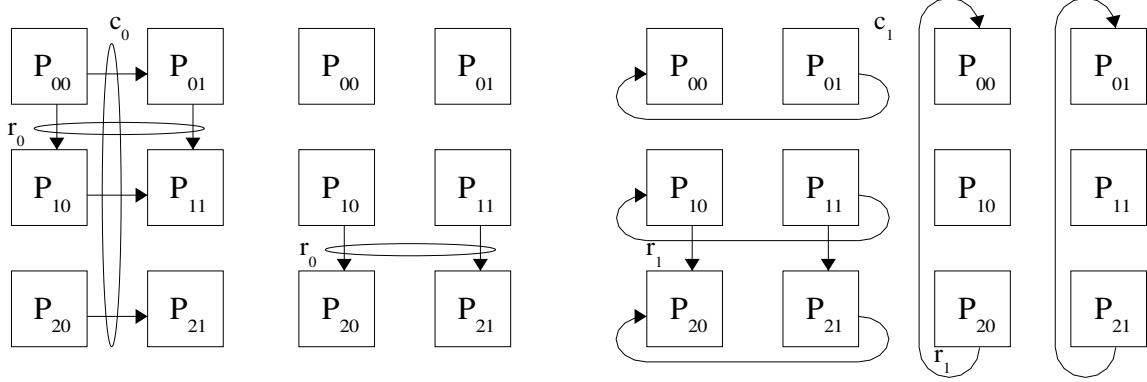
which resembles Cannon's algorithm.

(3) Focusing on delays produced by the effect of pipelining (note the time at which the first column of processors could send the first message of it's second and third column block), the concept of LCM(P, Q) (Least Common Multiple) is used in DIMMA (Distribution-Independent MMA) and P/Q ratio. DIMMA also defines explicit algorithmic rearrangements depending on k_{opt} and k_b (which is the first one in *recognizing* there is a difference).

An this algorithm is used in ScaLAPACK (PBLAS, more specifically).

10   Trabajos, Areas Abiertas

Para Empezar por alguna parte, material introductorio:

- Anderson T., D. Culler, D. Patterson, and the NOW Team, "A Case for Networks of Workstations: NOW", IEEE Micro, Feb. 1995.

- R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, 1999.

- R. Buyya Ed., High Performance Cluster Computing: Programming an Applications, Vol. 2, Prentice-Hall, Upper Saddle River, NJ, USA, 1999.

Específicamente para los trabajos finales del curso:

- Los que ya tienen proyecto de procesamiento paralelo/distribuido que involucra clusters: dimensionar un subproyecto.

- En todos los casos: un subproyecto. Algo que funcione y que se pueda medir/evaluar desde algún punto de vista.

- Areas abiertas de investigación.

10.1   Posibilidades

Varios niveles de abstracción/complejidad en varios niveles de los clusters como computadoras paralelas:
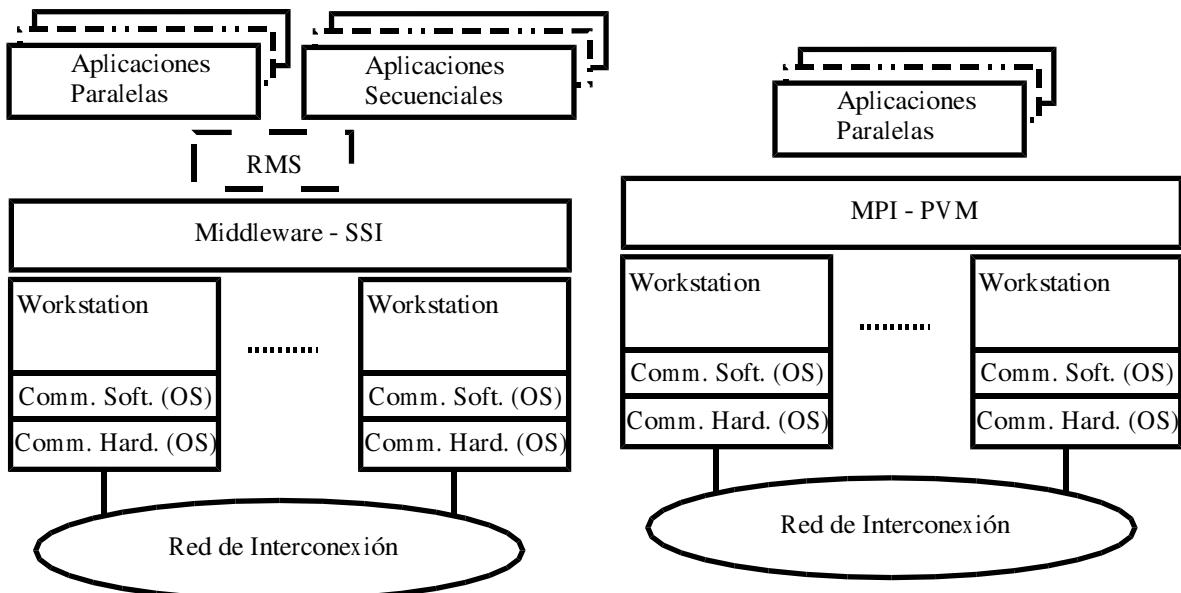
- A nivel de RMS: Condor, manejadores de colas, sistemas operativos "para clusters", etc.

- A nivel de middleware-SSI: sistemas operativos "para clusters", quizás algún aspecto de DSM, etc.

- High Availability - Replicación: quizás RMs, quizás SSI, etc.

- A nivel de aplicaciones paralelas: desde herramientas de debugging hasta paralelización de aplicaciones.

- A nivel de comunicaciones: muchas cosas, con bastante complejidad y niveles de abstracción.

Algunos temas relativamente genéricos, sobre los cuales se pueden definir algunos proyectos y/o subproyectos:

- Condor.

- OSCAR.

- Evaluación de Rendimiento: Linpack.

- Debugging y Optimización de una Aplicación Paralela.

- Específico: migración de procesos.

- Otros.

Sobre Condor:

- Descripción de laDocumentación.

- Instalación.

- Ejemplo de utilización/aprovechamiento.

- ¿Trabajos paralelos?

- ¿Monitorización (carga) de la red?

- Demostración de migración.

- Otras características.

Sobre OSCAR (Open Source Cluster Application Resources):

- http://oscar.sourceforge.net/

- oscar-users@lists.sourceforge.net

- The Open Cluster Group http://www.openclustergroup.org/

- Objetivos - Instalación.

- Instalación.

- Ventajas - Desventajas.

- Requerimientos (hardware - software).

- Recursos que maneja y cómo.

Sobre Evaluación de Rendimiento: Linpack

- Descripción de la Documentación.

- Instalación.

- Funcionamiento.

- Analizar sobre uno o más clusters.

- Comparar con TOP500.

- Como para enviar a TOP500.

Sobre Debugging y Optimización de una Aplicación Paralela: XMPI

- http://www.lam-mpi.org/software/xmpi/

- Descripción de objetivos.

- ¿Trazas, breakpoints, variables?

- Ejemplo de uso - Manual del usuario.

- Metodología de debugging y optimización (ej.).

- Evaluar overhead.

Sobre Checkpoint and Restart:

- En entorno Linux.

- Ejemplo de Condor.

- Restricciones.

- Biblioteca - Runtime.

- Ejemplo de instalación - utilización.

- Tolerancia a fallas-restart de computadoras.

- Manual de referencia - usuario.

Sobre Intercluster:

- Posibilidades.

- Motivaciones.

- Contexto de propuestas.

- ¡Hagamos algo!

Sobre Comunicaciones:

- Overhead de capas (software).

- Acknowledgements (software).

- Hardware.

- Otros.