A Comment on Block processing from the Practice with Matrix Multiplication

Remember the basic idea of block processing, specifically applied to multiply two 4×4 matrices: A, B, and C, are subdivided into 2×2 blocks ($bs = 2$). Each block/submatrix will have the *corresponding* row and column indexes

$$\begin{array}{|cc|cc|} \multicolumn{2}{c}{\mathrm{C}_{00}} & \multicolumn{2}{c}{\mathrm{C}_{01}} \\ \hline c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ \hline c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \\ \hline \multicolumn{2}{c}{\mathrm{C}_{10}} & \multicolumn{2}{c}{\mathrm{C}_{11}} \end{array} = \begin{array}{|cc|cc|} \multicolumn{2}{c}{\mathrm{A}_{00}} & \multicolumn{2}{c}{\mathrm{A}_{01}} \\ \hline a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ \hline \multicolumn{2}{c}{\mathrm{A}_{10}} & \multicolumn{2}{c}{\mathrm{A}_{11}} \end{array} \mathrm{x} \begin{array}{|cc|cc|} \multicolumn{2}{c}{\mathrm{B}_{00}} & \multicolumn{2}{c}{\mathrm{B}_{01}} \\ \hline b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ \hline b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \\ \hline \multicolumn{2}{c}{\mathrm{B}_{10}} & \multicolumn{2}{c}{\mathrm{B}_{11}} \end{array}$$

Partial Result: $\mathrm{C0}_{00} = \mathrm{A}_{00}$ x B_{00}
Partial Result: $\mathrm{C1}_{00} = \mathrm{A}_{01}$ x B_{10}

$$\mathrm{C0}_{00} = \begin{array}{|cc|} \hline a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \\ \hline \end{array}$$

$$\mathrm{C1}_{00} = \begin{array}{|cc|} \hline a_{02}b_{20} + a_{03}b_{30} & a_{02}b_{21} + a_{03}b_{31} \\ a_{12}b_{20} + a_{13}b_{30} & a_{12}b_{21} + a_{13}b_{31} \\ \hline \end{array}$$

Thus, $\mathrm{C0}_{00} + \mathrm{C1}_{00}$ should be C_{00}, i.e. the four elements at the *upper left* corner of C...

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

And the idea is true in general. But computing is not alone in cluster parallel (*high performance*) computing ...

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 x 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* using/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).
- 10/100 cost (by 2003-2004). Up to 8 computers: as a hub. 9 to 24 computers: as three hubs. More than 24: depends... Hint: switching cost grows more than linear.
- 10/100/1000 cost (by 2007). Up to 8 computers: 50-200 U$S. From 9 to 16: 100-250 U$S. 24: $>$ U$S 1500. 48: several thousands...
- Cascades: difficult to model performance.

Alternatives: myrinet, infiniband, quadrics, etc.

5 MPI: Message Passing Interface

Having into account the previous view of a cluster (pp. 15-16), something is needed in order to program and execute parallel applications. Initially, the *de facto* standard was PVM (Parallel Virtual Machine), http://www.csm.ornl.gov/pvm/, which is now considered *deprecated*, even when every year there is an Euro PVM/MPI conference.

The Message Passing Interface was defined in order to provide a standard programming model for distributed as well as shared memory parallel computers. "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/

The MPI standard defines a (*rather big*) set of routines which can be seen *initially* (in its MPI-1 "version") as implementing the CSP (Communicating Sequential Processes) model. With the definition of MPI-2, MPI includes ideas unrelated to CSP, such as remote memory access operations, and dynamic process management. It is always impressive to see how many *communication* subroutines/functions can be added to send/recv.

MPI is a standard definition, some implementation/s should be available at the cluster where you want to develop/run parallel HPC applications. Some known/*famous* implementations:

- MPICH: mostly based on TCP/IP, but the communication "device" used for/in the implementation can be changed (*ports*). *Almost* open source, or at least available in source code. Some web sites, http://www-unix.mcs.anl.gov/mpi/mpich1/, http://www.mcs.anl.gov/research/projects/mpich2/
- LAM/MPI: very similar to MPICH, and with some characteristics of PVM. Implementation mostly based on TCP/IP. Now (in fact since 2004) *moving* to Open MPI. Web sites http://www.lam-mpi.org/, http://www.open-mpi.org/
- Almost every hardware/software vendor for HPC has it own MPI implementation (Sun: Sun Cluster Tools, Intel: Intel MPI, etc.).
- Almost every HPC network communication hardware/fabric vendor also has its own MPI implementation (Myricom-Myrinet MPICH-MX, Infiniband MVAPICH, etc.).

MPI Parallel Program: set of independent processes for which MPI provides identification (*process ID*) and communication routines.

Tutorials: almost every national lab have one on its web site (Lawrence Livermore, Argonne, Oak Ridge, etc.). One of them, https://computing.llnl.gov/tutorials/mpi/

A Comment on α and β and Granularity from the Practice with MPI

Specific/*real* values for α and β are useful for *rough*/initial estimation of granularity: the computing/comunication ratio or an *a priori* estimation of possible parallel efficiency.

Example/experimental values found on Ethernet 100 MB/s, P4 (with a very basic MPI *pingpong*):

- $\alpha \cong 75\mu s = 75 \times 10^{-6} s$ (round-trip time of 1 byte message $\cong 150\mu s$).
- $\beta \cong 1/(11 \times 10^6) s \cong 10^{-7} s$ ($bndw \cong$ 11 MB/s for 1 MB messages).

Also, experiments with block matrix multiplication reported a performance of about 600 Mflop/s for $n = 1000$ and $bs = 50$. Now, some numbers/evaluation:

$$\begin{aligned} 50 \times 50 \; block &\Rightarrow 20000 \; bytes \\ t_{comm}(block) &\cong 75 \times 10^{-6} \; s \; + 20000 \;\; 10^{-7} \; s = (75 + 2000) \times 10^{-6} \; s \\ &\cong 2 \times 10^{-3} \; s \end{aligned}$$

Relating communication time with flops, or flops required by 50×50 blocks multiplication in 600 Mflop/s computers (6×10^8 flop/s)

$$\begin{aligned} Flops \; required &\cong 50^3 = 125000 = 1.25 \times 10^5 \\ t_{comp}(block) &\cong \frac{1.25 \times 10^5}{6 \times 10^8} \; s = \frac{1.25}{6} \times 10^{-3} \; s \cong 0.2083 \times 10^{-3} \; s \\ &\cong 2 \times 10^{-4} \; s \end{aligned}$$

Roughly (note multiple $\cong$),

$$\frac{t_{comm}(block)}{t_{comp}(block)} = \frac{2 \times 10^{-3}}{2 \times 10^{-4}} = 10$$

i.e. the communication time of a block is one order of magnitude greater than a single usage (matrix multiplication) of that block. One of the corollaries: using 10 times a block in a parallel program is needed for using just 50% of the available computing power in a parallel program following the sequence: communicate-compute. The number of times a block can be re/used in partial computations depends of many factors in a parallel program, being data distribution and number of computers the two most important ones.