# Efficiency and scalability on computing cluster - can it be achieved ?

Sławomir Potempski

DUZ/UZ3

National Centre for Nuclear Research

# Content

- Efficiency and scalability of computer systems
- Analysis of execution time of parallel algorithms
- Example: system of linear equations with tridiagonal matrix derived from transport equation
- Other applications

# CPU time

MFLOPS – number of millions floating point operations per second

MFLOPS(A)= $n_{flp\_op}(A)/(T_{U\_CPU}(A) \bullet 10^6)$

$T_{U\_CPU}(A)$ – user time CPU for instruction $A$

$n_{flp\_op}(A)$ – number of floating point instructions $A$

MFLOPS – does not distinguish different floating point operations (e.g. sum vs square root), however it is convenient for comparing various versions of the program implementation of the same computing algorithm.

# CPU time

Taking into account memory operations:

$$T_{\text{U\_CPU}}(A) = (n_{\text{cycle}}(A) + n_{mm\_\text{cycle}}(A)) \bullet t_{\text{cycle}}$$

$$n_{\text{mm\_cycles}}(A) = n_{\text{read\_cycles}}(A) + n_{\text{write\_cycles}}(A)$$

$$n_{\text{read\_cycles}}(A) = n_{\text{read\_op}}(A) \bullet r_{\text{read\_miss\_rate}}(A) \bullet n_{\text{miss\_cycle}}$$

$$T_{\text{U\_CPU}}(A) = n_{\text{instr}(A)} \bullet [CPI\ (A) + n_{\text{rw\_op}}(A) \bullet r_{\text{miss\_rate}}(A) \bullet n_{\text{miss\_cycle}}] \bullet t_{\text{cycle}}$$

CPI($A$) - clock cycles per instruction $A$

Average access time to memory:

$$t_{\text{read\_access}}(A) = t_{\text{read\_hit}} + r_{\text{read\_miss\_rate}}(A) \bullet t_{\text{read\_miss}}$$

Further consideration can be done for different levels of cache memories.

# Measures of efficiency of parallel programs

$T_p(n)$ – execution time of parallel program of size $n$ on $p$ processors = time from the start till the end of all tasks running on all processors, containing:

- Time of local computation
- Time of data exchange between processors
- Synchronization time
- Waiting time

# Measures of efficiency of parallel programs

Cost of execution parallel program

$C_p(n)=pT_p(n)$ – measure of total work

Cost is optimal when: $C_p(n)=T_1(n)$ (time of the fastest sequential program)

Speedup= execution time of sequential program /execution time of parallel program

Efficiency = cost of sequential program /cost of parallel program

$$E_p = \frac{C_1}{C_p} = \frac{T_1}{pT_p} \quad S_p = \frac{T_1}{T_p} = pE_p \leq p$$

In practice superlinear speedup can be observed ($S_p > p$)

# Amdahl's and Gustafsion's laws

**Amdahl's law**:

If for some problem, serial part is *s* (*0*≤*s*≤*1*), and *1*-*s* is realized parallelly on *p* processes then:

$$E_p = \frac{T_1}{pT_p} = \frac{1}{sp + (1-s)} \underset{p\to\infty}{\to} 0 \qquad S_p = \frac{T_1}{T_p} = \frac{p}{sp + (1-s)} \underset{p\to\infty}{\to} \frac{1}{s}$$

For example if *s*>10% then $S_p < 10$

**Gustafson's law**: Speedup is scalable for enough big problems:

$$S_p = \frac{t_s + t_p(n,1)}{t_s + t_p(n,p)} = \frac{t_s + T_1(n) - t_s}{t_s + (T_1(n) - t_s)/p} = \frac{\frac{t_s}{T_1(n) - t_s} + 1}{\frac{t_s}{T_1(n) - t_s} + \frac{1}{p}} \underset{n\to\infty}{\to} p$$

$t_s$ – time of serial part, $t_p$ – time of parallel part

**WIKIPEDIA – a driving metaphor**

Amdahl's law:

- Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph. No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city. Since it has already taken you 1 hour and you only have a distance of 60 miles total; going infinitely fast you would only achieve 60 mph.

Gustafson's law:

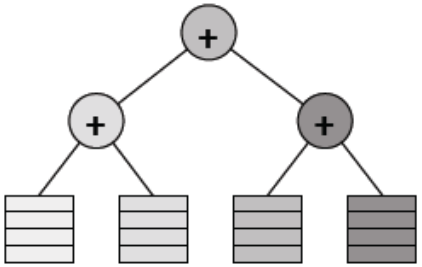
- Suppose a car has already been traveling for some time at less than 90mph. Given enough time and distance to travel, the car's average speed can always eventually reach 90mph, no matter how long or how slowly it has already traveled. For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour, and so on.

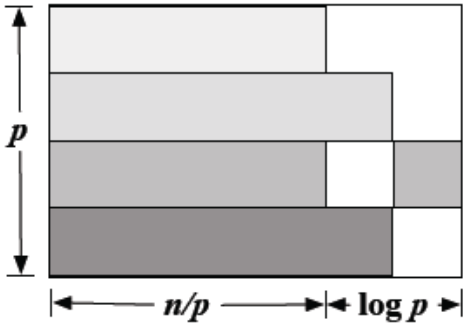
# Scalability

Algorithm is scalable if its efficiency is of constant order along with the increase of the number of processors i.e. $E_p = O(1)$ gdy $p \to \infty$

Example: summing is not scalable

$$E_p = \frac{n}{n + p \log p} \underset{p \to \infty}{\to} 0$$




*Notation:*

$$O(g(n)) = f(n) \Leftrightarrow \exists c > 0, 0 \le f(n) \le c(g(n))$$

$$\Omega(g(n)) = f(n) \Leftrightarrow \exists c > 0, 0 \le c\, g(n) \le f(n)$$

$$\Theta(g(n)) = f(n) \Leftrightarrow \exists c_1, c_2 > 0, c_1 g(n) \le f(n) \le c_2 g(n)$$

# Parallel efficiency

- Efficiency: effectiveness of parallel algorithm in comaprison with serial processing
- Load balancing: distribution of tasks among processors
- Concurrency: set of processors functioning simultaneously
- Overhead: additional work not existing in sequential algorithm

Efficiency is maximal when: load balancing and concurrency are maximal and overhead is minimal

# Efficiency and scalability

Execution time = (total work)/(velocity)

– sequential: $T_1 = W_1/V(M)$

– parallel: $T_p = W_p/(pV(M/p))$

$W$–work, $M$-memory, $V$-velocity (flops)

Parallel overhead: $W_p - W_1$

$W_1$ is a function $W_1(n)$, where parameter $n$ characterizes the problem, assuming $W_1(n) - EW_p(n,p) = 0$ for some constant $E$, this dependency defines (implicitly) $n$ as a function of $p$.

# Efficiency and scalability

Dependency $W_1(n)=EW_p(n,p)$, defines $n(p)$.
$n$ defines the characteristics of the problem – for example multiplying two matrices leads to $W_1(n)=O(n^3)$

$W_1(n(p))$ - isoefficiency function.

In case of summation:

$W_1 = EW_p$ $n \sim E(n + p \log p)$, hence:

$n = \Theta(p \log p)$ , $W_1(n,p) = \Theta(p \log p)$

i.e. if the size icreases like ~ $p \log p$, then the algorithm is scalable, however execution time grows as $\log p$.

# Efficiency and scalability

In general: $T_p = W_1/(pE)$ is constant if isoefficiency function is of order $\Theta(p)$, otherwise $T_p$ grows as $p$ grows.

Isoefficiency of order $O(p)$, $O(plogp)$, $O(p^{3/2})$ is desirable, while isoefficiency > $O(p^2)$ means weak sclability because, $T_p$ grows faster than linear with the increased number of processors $p$.

Isoefficiency of order $O(p)$ for many problems cannot be reached.

# Example: linear transport equation

1-D transport equation: $u_t(t,x) + \alpha u_x(t,x) = 0$

Grid: $\{t_n, x_k\}$, $t_n = n\tau$, $x_k = kh$.

Integrating over $[x_k, x_{k+1}] \times [t_n, t_{n+1}]$

$$\int_{x_k}^{x_{k+1}} \int_{t_n}^{t_{n+1}} u_t dt dx + \alpha \int_{t_n}^{t_{n+1}} \int_{x_k}^{x_{k+1}} u_x dx dt = 0$$

and using trapezoidal rule for integrals we get („box scheme”)”:

$$\frac{h}{2}[(u_k^{n+1} + u_{k+1}^{n+1}) - (u_k^n + u_{k+1}^n)] + \frac{\alpha\tau}{2}[(u_{k+1}^n + u_{k+1}^{n+1}) - (u_k^n + u_k^{n+1})] = 0$$

$$a u_{k+1}^{n+1} + b u_k^{n+1} = b u_{k+1}^n + a u_k^n, \quad a = 1 + \lambda\alpha, \; b = 1 - \lambda\alpha. \qquad \lambda = \frac{\tau}{h}$$

# System of linear equations

$$\begin{bmatrix} a & & & & & \\ b & a & & & & \\ & b & a & & & \\ & & & \ddots & & \\ & & & b & a & \\ & & & & b & a \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \\ \\ \\ x_M \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \\ \\ \\ f_M \end{bmatrix}$$

$$\left[\begin{array}{ccccc} D & & & & \\ \hline A & D & & & \\ \hline & A & D & & \\ \hline & & A & D & \\ \hline & & & A & D \end{array}\right] \begin{bmatrix} \underline{X_0} \\ \underline{X_1} \\ \underline{X_2} \\ \underline{X_3} \\ \underline{X_4} \end{bmatrix} = \begin{bmatrix} \underline{F_0} \\ \underline{F_1} \\ \underline{F_2} \\ \underline{F_3} \\ \underline{F_4} \end{bmatrix}$$

# System with tridiagonal matrix

$$\begin{bmatrix} d_0 & b_0 & & & & & \\ a_1 & d_1 & b_1 & & & & \\ . & . & . & . & . & . & . \\ & & & & & a_M & d_M \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \dots. \\ x_M \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \dots. \\ f_M \end{bmatrix}$$

Thomas algorithm, progonka – version of Gauss elimination

Solution is based on the formulas:

$x_j = \propto_j x_{j+1} + \beta_j$

$$\propto_j = -\frac{b_j}{a_j \propto_{j+1} + d_j} \qquad \propto_0 = -\frac{b_0}{d_0}$$

$$\beta_j = -\frac{f_j - \beta_{j-1}}{a_j \propto_{j-1} + d_j} \qquad \beta_0 = \frac{f_0}{d_0}$$

$$x_M = \beta_M$$

Algorithm is essentially sequential with complexity *O(M)*

# System with tridiagonal matrix

$$\begin{bmatrix} d_0 & b_0 & & & & & \\ a_1 & d_1 & b_1 & & & & \\ . & . & . & . & . & . & . \\ & & & & & a_M & d_M \end{bmatrix}$$

$M=l_p(R+1)-1$

$l_p$- number of processors

$A_s, D_s, B_s \sim (R+1)x(R+1)$

$$\left[\begin{array}{c|c|c|c|c} D_0 & B_0 & . & . & . \\ \hline A_1 & D_1 & B_1 & . & . \\ \hline . & A_2 & D_2 & B_2 & . \\ \hline . & . & A_3 & D_3 & B_3 \\ \hline . & . & . & A_4 & D_4 \end{array}\right] \begin{bmatrix} \underline{X_0} \\ \underline{X_1} \\ \underline{X_2} \\ \underline{X_3} \\ X_4 \end{bmatrix} = \begin{bmatrix} \underline{F_0} \\ \underline{F_1} \\ \underline{F_2} \\ \underline{F_3} \\ F_4 \end{bmatrix}$$

$$X_0 = \tilde{X}_0 - D_0^{-1} B_0 X_1$$

$$X_1 = \tilde{X}_1 - D_1^{-1} A_1 X_0 - D_1^{-1} B_1 X_2$$

$$X_2 = \tilde{X}_2 - D_2^{-1} A_2 X_1 - D_2^{-1} B_2 X_3$$

$$X_3 = \tilde{X}_3 - D_3^{-1} A_3 X_2 - D_3^{-1} B_3 X_4$$

$$X_4 = \tilde{X}_4 - D_4^{-1} A_4 X_3$$

$$\widetilde{X_s} = D_s^{-1} F_s,$$

$$A_s = a_{s(R+1)} e_0 e_R^T,$$

$$B_s = b_{(s+1)R+s} e_R e_0^T,$$

$e_i$ - versor

# System with tridiagonal matrix

By multipying first $l_p$ -*1* equations by $e^T_R$ and last $l_p$ -*1* by $e^T_0$ we get Schur system of size 2($l_p$-*1)*

Even unknowns

$$Z_0 = e_0^T X_1$$
$$Z_2 = e_0^T X_2$$
$$Z_4 = e_0^T X_3$$
$$Z_6 = e_0^T X_4$$
$$\cdots$$
$$Z_{2(lp-1)-2} = e_0^T X_{lp-1}$$

Odd unknowns

$$Z_1 = e_R^T X_0$$
$$Z_3 = e_R^T X_1$$
$$Z_5 = e_R^T X_2$$
$$Z_7 = e_R^T X_3$$
$$\cdots$$
$$Z_{2(l_p-1)+1} = e_R^T X_{lp-2}$$

$$b^0 e_R^T V^0 z_0 + z_1 = e_R^T \tilde{X}_0$$

$$z_0 + a^1 e_0^T W^1 z_1 + b^1 e_0^T V^1 z_2 = e_0^T \tilde{X}_1$$

$$a^1 e_R^T W^1 z_1 + b^1 e_R^T V^1 z_2 + z_3 = e_R^T \tilde{X}_1$$

$$z_2 + a^2 e_0^T W^2 z_3 + b^2 e_0^T V_2 z_4 = e_0^T \tilde{X}_2$$

$$a^2 e_R^T W^2 z_3 + b^2 e_R^T V^2 z_4 + z_5 = e_R^T \tilde{X}_2$$

$$z_4 + a^3 e_R^T W^3 z_5 = e_0^T \tilde{X}_3$$

$$W^s = D_s^{-1} e_0 = [w_0^s, w_1^s, \cdots, w_R^s]^T,$$

$$V^s = D_s^{-1} e_R = [v_0^s, v_1^s, \cdots, v_R^s]^T.$$

# Algorithm (S. Bondelli)

1. Finding $\widetilde{X_s}$ and vectors $V_s$, $W_s$. This can be done by solving system with tridiagonal matrices $D_s$ and right hand sides: $F_s$, $e_0$, $e_R$. Typical Thomas algorithm can be used („progonka”).

2. Exchange of information such that in each processor all the coefficients and right hand sides are available (*MPI_Allgather*).

3. In each processor Schur system is solved

# Algorithm (S. Bondelli)

4. Then the following equations produces solution, such that at processor $k$ $X_k$ is available.

$$X_0 = \tilde{X}_0 - b_R V^0 Z_0$$

$$X_1 = \tilde{X}_1 - a_{R+1} W^1 Z_1 - b_{2R+1} V^1 Z_2$$

$$X_2 = \tilde{X}_2 - a_{2(R+1)} W^2 Z_3 - b_{3R+2} V^2 Z_4$$

$$X_3 = \tilde{X}_3 - a_{3(R+1)} W^3 Z_5 - b_{4R+3} V^3 Z_6$$

$$X_4 = \tilde{X}_4 - a_{4(R+1)} W^4 Z_7$$

As all the steps in the algorithm have complexity linear with respect to the size of the problem and number of processors, then isoefficiency is of order *O(p)*.

# Some important differential equations

- Transport equation:

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - \nabla \cdot (\vec{v} c) + R,$$

where $\vec{v}$ - velocity, $R$ – source, $c$ – variable of interest

- Conservation law:

$$\frac{\partial u}{\partial t} + div\big(F(u)\big) = 0$$

- Navier-Stokes equations:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \Delta u = -\nabla p$$

$$divu = 0$$

- Heat equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u) + f(x, t)$$

# Application to heat equation

ADI – Alternative Direction Implicit method Peaceman-Rachfort for 2D heat equation

- $\frac{y_{i,j}^{n+1/2} - y_{i,j}^{n}}{\tau/2} = \frac{D}{2}\{\delta_x^2 y_{i,j}^{n+\frac{1}{2}} + \delta_y^2 y_{i,j}^{n}\}$
- $\frac{y_{i,j}^{n+1} - y_{i,j}^{n}}{\tau/2} = \frac{D}{2}\{\delta_x^2 y_{i,j}^{n+\frac{1}{2}} + \delta_y^2 y_{i,j}^{n+1}\}$

where $\delta_x$, $\delta_y$ = finite differences with respect to $x$,$y$.

This leads to the system of linear equations with tridiagonal matrices. Extension to 3D is straightforward. This concept can be generalized for some „splitting" numerical schemes.

# Splitting techniques

- Example: 2D transport equation: $u_t + \alpha_1 u_{x_1} + \alpha_2 u_{x_2} = 0$

$$v_t + \alpha_1 v_{x_1} = 0, \quad v(t_n, \cdot) = u(t_n, \cdot)$$

$$w_t + \alpha_2 w_{x_2} = 0, \quad w(t_n, \cdot) = v(t_{n+1}, \cdot)$$

- Example: 2D convection-diffusion equation:

$$u_t + \underline{\alpha} \nabla u - \nu \Delta u = 0$$

Transport equation for $u_1$, $u_2$

$$u_{1,t} + \alpha_1 u_{1,x_1} = 0, \quad u_1(t_n, \cdot) = u(t_n, \cdot)$$

$$u_{2,t} + \alpha_2 u_{2,x_2} = 0, \quad u_2(t_n, \cdot) = u_1(t_{n+1}, \cdot)$$

Heat equation for $u_3$, $u_4$

$$u_{3,t} - \nu u_{3,x_1,x_1} = 0, \quad u_3(t_n, \cdot) = u_2(t_{n+1}, \cdot)$$

$$u_{4,t} - \nu u_{4,x_2,x_2} = 0, \quad u_4(t_n, \cdot) = u_3(t_{n+1}, \cdot)$$

# The Stokes problem

The Stokes equation describes the motion of an incompressible viscous flow in n-dimensional domain (n=2,3): find u – velocity and p – pressure such that:

$$-\Delta u - gradp = f \quad in\ \Omega \qquad u : \Omega \to R^n,$$

$$divu = 0 \quad in\ \Omega \qquad p : \Omega \to R$$

$$u = u_0 \ on\ \partial\Omega$$

Applying finite element, finite difference or finite volume method, finally we get the system of linear equations with specific form.

# Algebraic system

Primal problem: $$S\begin{bmatrix}\underline{u}\\ \underline{p}\end{bmatrix}=\begin{bmatrix}A_h & B_h^T\\ B_h & 0\end{bmatrix}\begin{bmatrix}\underline{u}\\ \underline{p}\end{bmatrix}=\begin{bmatrix}\underline{f}\\ \underline{g}\end{bmatrix}$$

Matrix $A_h$ is symmetric positive definite
Matrix $B_h{}^T$ has full column rank
Matrix $B_h A_h{}^{-1} B_h{}^T$ is symmetric positive definite
Matrix $S$ is symmetric invertible non-definite
(it has $N$ positive and $M$ negative eigenvalues)

Instead of solving primary problem with matrix $S$ it's better to solve the following dual problem:

$$B_h A_h^{-1} B_h^T \underline{p} = B_h A_h^{-1} \underline{f} - \underline{g}$$

$$A_h \underline{u} = \underline{f} - B_h^T \underline{p}$$

# Variational formulation of stationary Navier-Stokes equations

Multiplying the first NS equation by smooth function *v* vanishing on boundary, integrating by parts, and multiplying the second NS equation by some function *q* we get formulation of the following form:

$$a(u,v)+b(u,u,v)-(divv,p)=(f,v) \quad \forall v\in X \quad (v,\nabla p)=-(divv,p)$$

$$(divu,q)=0 \quad \forall q\in M$$

$$a(u,v)=\frac{1}{\mathrm{Re}}\int_\Omega \nabla u\cdot\nabla v dx \quad b(u,v,w)=\int_\Omega (u\cdot\nabla v)\cdot w dx=\int_\Omega \sum_{i,j=1}^{n} u_j \frac{\partial v_i}{\partial x_j} w_i dx$$

$$a: X\times X\to R \quad b: X\times X\times X\to R$$

$$X=(H_0^1(\Omega))^n \quad M=L^2(\Omega)/R$$

# Approximation of stationary Navier-Stokes equations

$$a(u_h, v_h) + b(u_h, u_h, v_h) - (div\, v_h, p_h) = (f, v_h) \quad \forall v_h \in X_h$$

$$(div\, u_h, q_h) = 0 \quad \forall q_h \in M_h$$

$$X_h \subset X \quad M_h \subset M$$

Approximation using the iteration of the Stokes problem

*Start with:* $(u_h^0,\ p_h^0)$

*Knowing* $(u_h^n,\ p_h^n)$ *find* $(u_h^{n+1},\ p_h^{n+1})$ *such that:*

$$a(u_h^{n+1}, v_h) + b(u_h^n, u_h^n, v_h) - (div\, v_h, p_h^{n+1}) = (f, v_h) \quad \forall v_h \in X_h$$

$$(div\, u_h^{n+1}, q_h) = 0 \quad \forall q_h \in M_h$$

# Schur complement

$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, $M \sim (p+q) \times (p+q)$, $A \sim p \times p$, $D \sim q \times q$

Schur complement: $A - BD^{-1}C \sim p \times p$

$$Ax + By = f$$
$$Cx + Dy = g$$

If both $D$ and Schur complement can be inverted, then the problem is reduced to invert two matrices of dimensions $q \times q$ and $p \times p$ respectively

$(A - BD^{-1}C)x = f - BD^{-1}g$

# Conclusions

- There is no straightforward way to transfer the code from sequential to parallel form
- Typically new numerical algorithms have to be developed to efficiently perform simulations on computing cluster
- It is worth to start with simpler problems, which can be further utilised for more complex ones - splitting techniques can be then applied