Introduction to parallel computing part I

Aslak Tveito

University of Oslo

Geilo, March 2001

Winter School in Scientific Computing

- Introduction
- Programming
- Domain decomposition
- Beowulf clusters
- Applications

This lecture

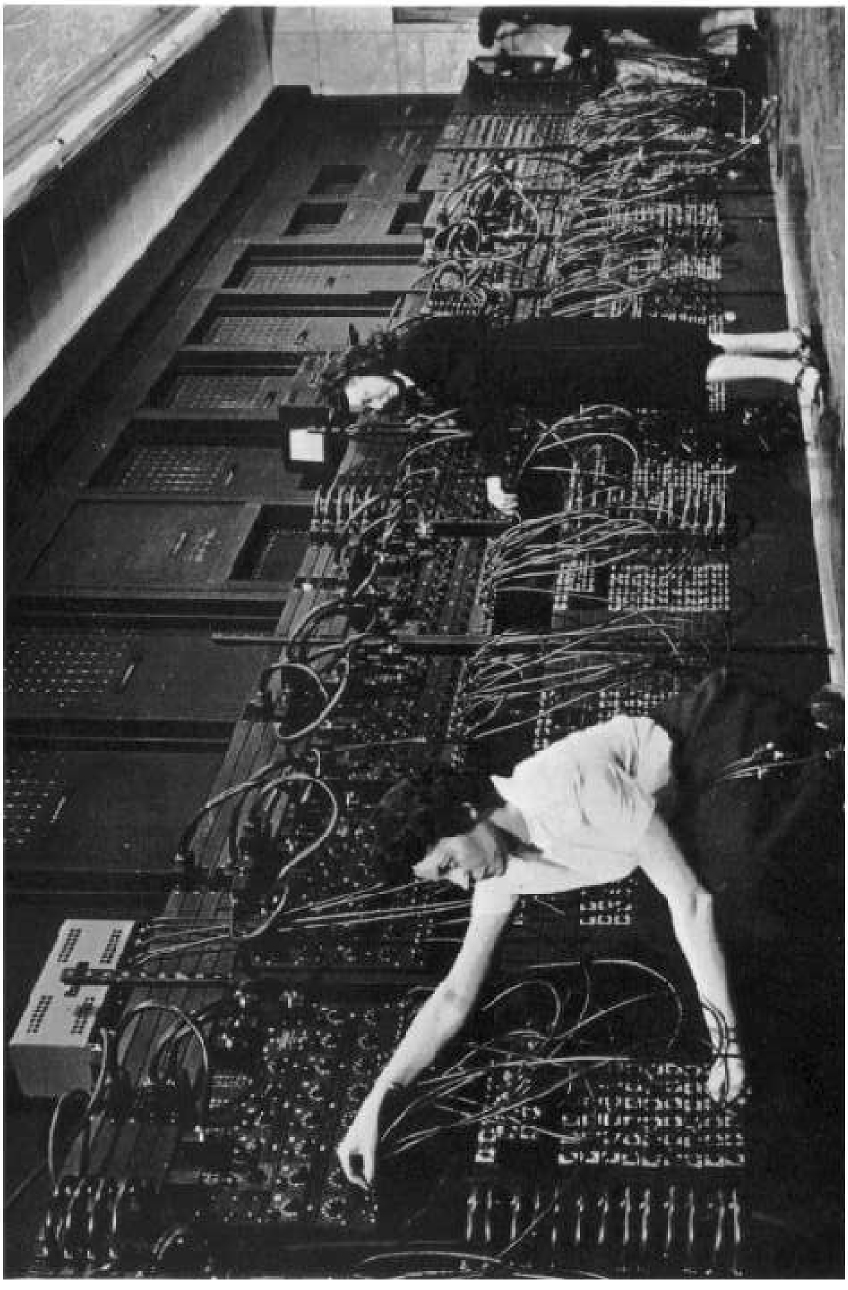
- Scientific computing
- Computers
- Model of a parallel computer
- Examples
- Performance analysis
- Partial Differential Equations
- Discretization
- Algorithms
- More on performance analysis

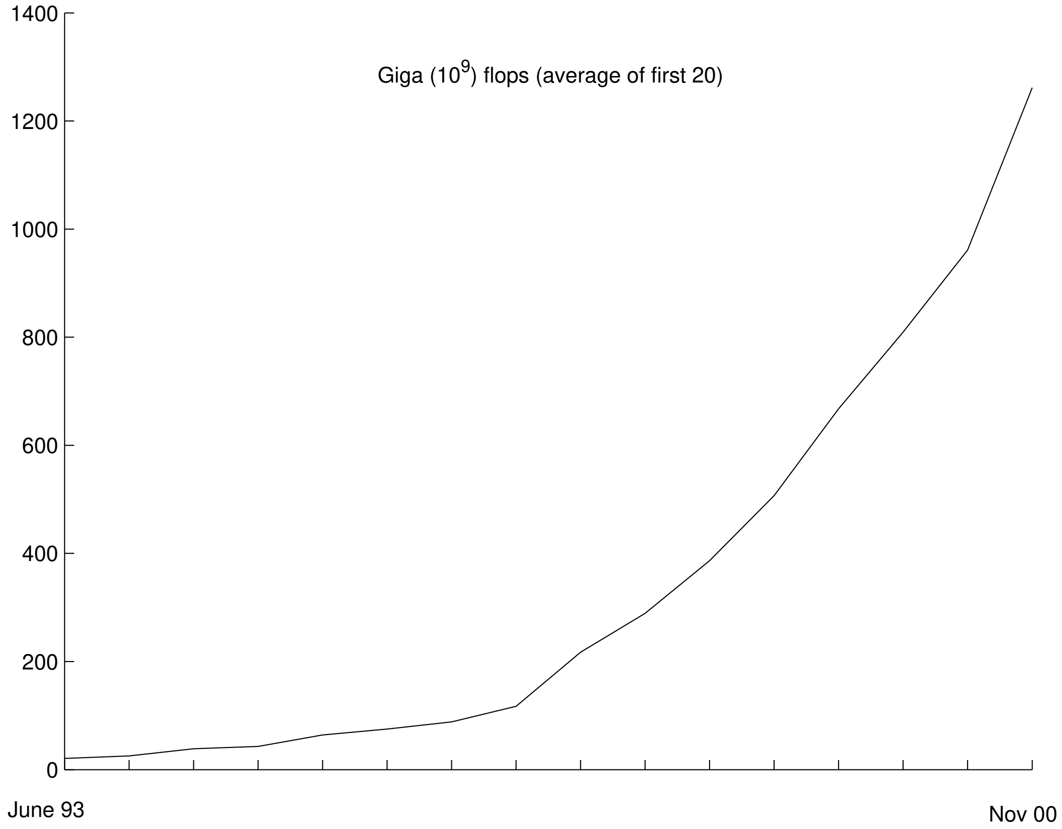
John von Neumann’s vision

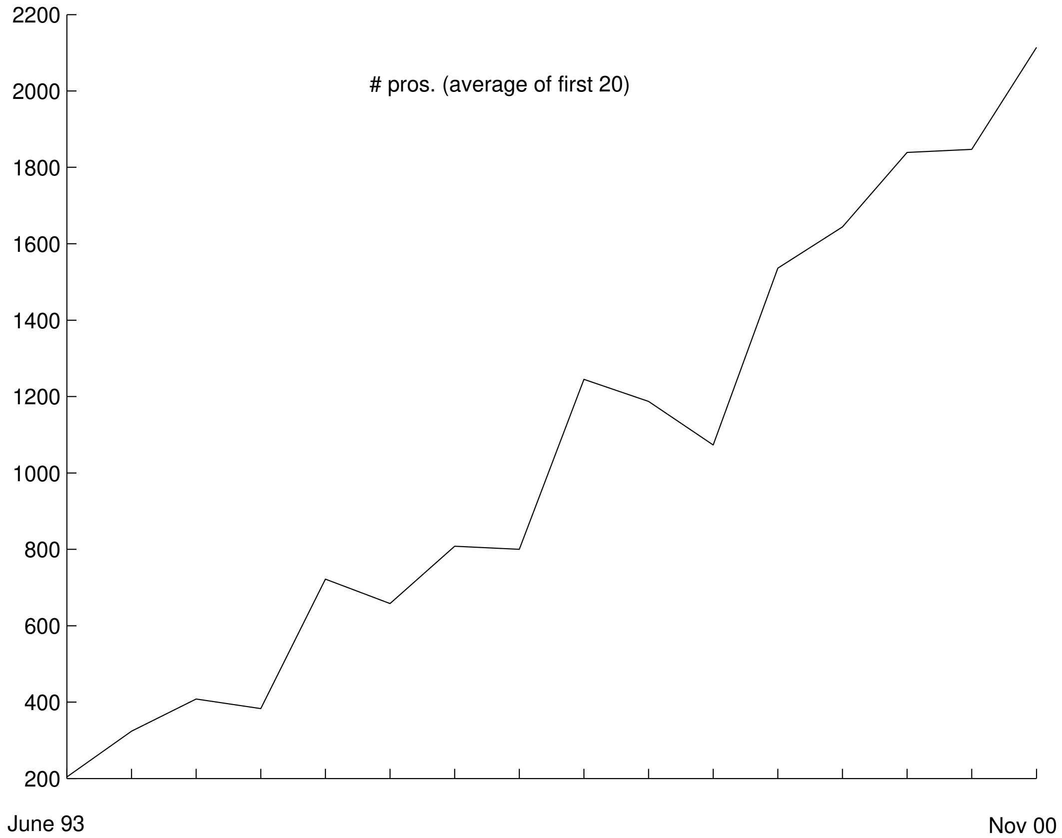
“Indeed, to a great extent, experiments in fluid mechanics are carried out under conditions where the underlying physical principles are not in doubt, where the quantities to be observed are completely determined by known equations. The purpose of the experiment is not to verify a proposed theory but to replace a computation from an unquestioned theory by direct measurements.”

ENIAC (1946): 330 FLOPs

Computer	Date Aquired	Storage (words)	Speed
IBM 604	Before 1950	8(?) + Cards	
IBM CPC	1952	56 + Cards	5 FLOPS
BENDIX G-15	1955	864	10 FLOPS
IBM 704	1956 (away)	8,192	10,000 FLOPS
BENDIX G-20	1961	8,192	20,000 FLOPS
IBM 7040/7044	1962	16,384	40,000 FLOPS
IBM 360/65	1967	256K	400,000 FLOPS
IBM 370/165	1971	1 M	1 MFLOPS
IBM 370/168	1975	2 M	1.2 MFLOPS
AMDAHL V8	1978	4 M	2 MFLOPS
IBM 3033	1979	4 M	2 MFLOPS
IBM 3081	1982	4 M	6 MFLOPS
CRAY 1S	1982	4 M	20-160 MFLOPS




Giga (10^9) flops (average of first 20)
1400
1200
1000
800
600
400
200
0
June 93
Nov 00

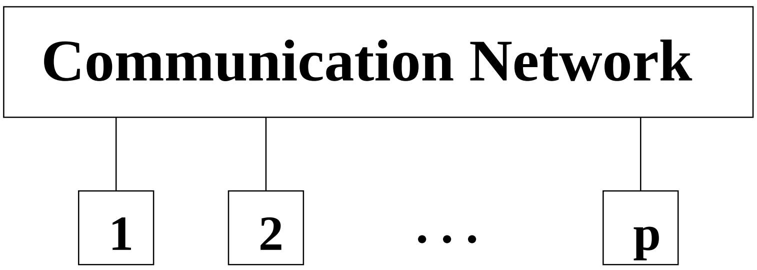


2200
2000
1800
1600
1400
1200
1000
800
600
400
200
pros. (average of first 20)
June 93
Nov 00

A model of a parallel computer

A multicomputer:

- p processors (nodes); $1, 2, \ldots, p$
- All processors are identical.
- All processors compute seqentially.
- All nodes can communicate with any other node.
- The communication is handled by mechanisms for sending and receiving data at each processor (send/receive).



Adding two vectors

Given

- vectors x, y with components $\{x_i, y_i\}_{i=1}^N$
- processors $1, 2, \ldots, p$
- integer n such that $N = np$

Problem: Compute

$$z = x + y$$

using p processors.

Step 1: Distribute x, y, z (allocate and store)

$$\underbrace{x_1, x_2, x_3, \ldots, x_n}_{\text{proc. } 1}, \underbrace{x_{n+1}, x_{n+2} \ldots, x_{2n}}_{\text{proc. } 2},$$
$$\ldots, \underbrace{x_{np-n+1}, x_{np-n+2} \ldots, x_{np}}_{\text{proc. } p}$$

Similarly for y and z.

Local and global numbering

Global numbering: $x_1, x_2, \ldots, x_N$.

Local numbering: On each processor we have $x_1, x_2, \ldots, x_n$.

There is a unique mapping between a global index j $(j = 1, \ldots, N)$ and a local index i $(i = 1, \ldots, n)$ at processor number q $(q = 1, \ldots, p)$.

Given a local index i at processor q, the corresponding global index is

$$\boxed{j = (q-1)n + i}$$

Similarly, given j, the processor number q is determined by requiring that

$$\boxed{(q-1)n+1 \leq j \leq qn}$$

and i is then given by

$$\boxed{i = j - (q-1)n}$$

Global	Local	Proc
x_1 x_2 $\vdots$ x_n	x_1 x_2 $\vdots$ x_n	proc. 1
x_{n+1} $\vdots$ x_{2n}	x_1 $\vdots$ x_n	proc. 2
$\vdots$	$\vdots$	$\vdots$

Step 2: On each processor, compute

$$z_i = x_i + y_i,\ i = 1, \ldots, n$$

Summary:

1. Distribute
2. Compute
3. (Gather)

Computing an inner product

Given

- vectors x, y with components $\{x_i, y_i\}_{i=1}^N$
- processors $1, 2, \ldots, p$
- integer n such that $N = np$

Problem: Compute

$$a = (x, y) = \sum_{i=1}^{N} x_i y_i$$

Step 1:

Distribute x and y to the p processors.

Step 2:

On each processor, compute

$$a(q) = \sum_{i=1}^{n} x_i y_i \text{ (q is the processor number)}$$

where x and y are local variables.

Step 3:

On each processor, send the result $a(q)$ to all other processors.

Step 4:

Receive $a(q)$ from each other processor.

Step 5:

On each processor, compute

$$a = \sum_{q=1}^{P} a(q)$$

Now $a = (x, y)$ is stored on each processor.

Performance analysis

Let p be the number of processors on a multicomputer, and let

$$T = T(p)$$

denote the execution time for a program on a multicomputer.

Performance analysis is the study of the properties of $T(p)$.

In order to analyze concurrent algorithms, we have to assume certain properties of the computer. In fact these assumptions are rather strict and thus leaving out a lot of existing computers. On the other hand; without these assumptions the analysis tend to be extremely complicated.

Observation

Let $T(1)$ be the fastest possible scalar computation, then

$$T(p) \geq \frac{T(1)}{p}.$$

This relation states a bound for how fast a computation can be done on a parallel computer compared with a scalar computation.

Definitions

In order to discuss aspects of performance analysis, we have to introduce some commonly used terms.

<u>Speed-up</u>

The speed-up of a $p-$node computation with execution time $T(p)$ is given by

$$S(p) = \frac{T(1)}{T(p)}$$

where $T(1)$ is the execution time of a scalar computation of the same problem and on the same computer.

<u>Efficiency</u>

The efficiency of a $p-$node computation with <u>speed-up</u> $S(p)$ is given by

$$\eta(p) = \frac{S(p)}{p}.$$

Discussion

Suppose that we are in an optimal situation, i.e. we have

$$T(p) = \frac{T(1)}{p}.$$

Then the speed-up is given by

$$S(p) = \frac{T(1)}{T(p)} = p,$$

and the efficiency is

$$\eta(p) = \frac{S(p)}{p} = 1.$$

More generally we have

$$T(p) \geq \frac{T(1)}{p},$$

which implies that

$$S(p) = \frac{T(1)}{T(p)} \leq p,$$

and

$$\eta(p) = \frac{S(p)}{p} \leq 1.$$

In practical computations we are very pleased if we are close to the optimal results. A speed up close to p and a an efficiency close to 1 is very good. Many annoying practical details often result in weaker results than what we expect from performance analysis.

Efficiency modelling

The goal of efficiency modelling is to estimate how fast a certain algorithm will run on a multicomputer. The models usually depends on the following parameters:

$\tau_A =$ Arithmetic time; the time of one single arithmetic operation. Integer arithmetics are usually ignored, and all nodes are assumed to be equal.

$\tau_C(L) =$ Message exchange time; the time it takes to send a message of length L (in proper units) from one processor to another. We assume that this time is equal for any two pairs of processors on our multicomputer.

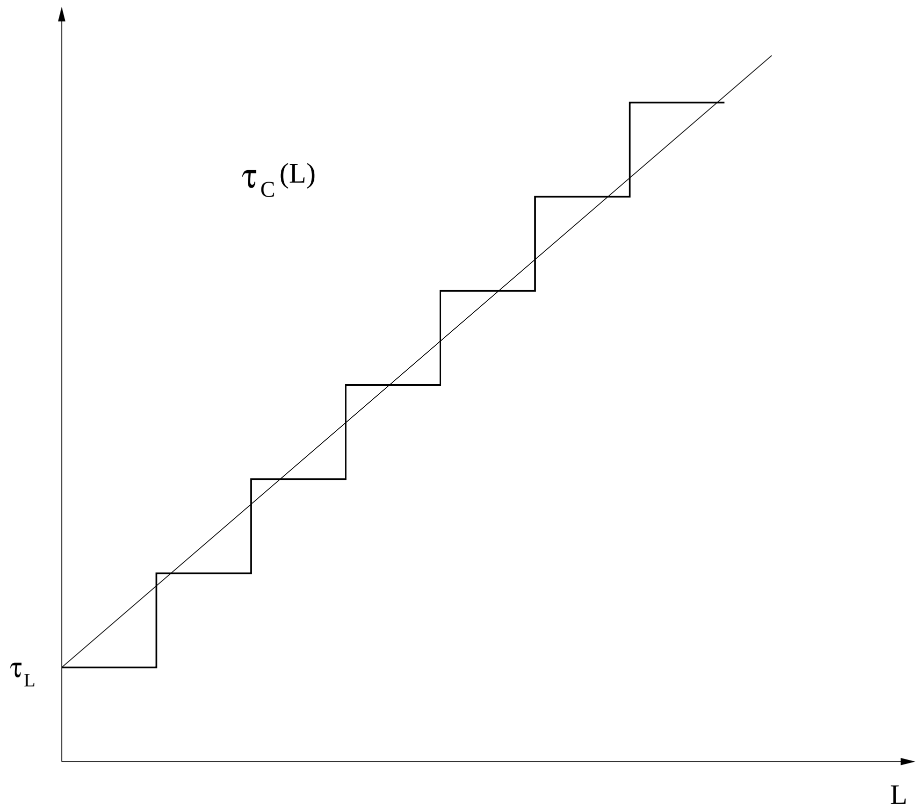
$\tau_L =$ Latency; the start-up time for a communication - or the time it takes to send a message of length zero.

$\frac{1}{\beta} =$ Band width; the maximum rate of messages (in proper units) that can be exchanged.

In our efficiency models, we will assume that there is a linear relation between the message exchange time and the length of the message:

$$\tau_C(L) = \tau_L + \beta L.$$

More realistic model



The function $\tau_C(L)$ is a piecewise constant function which can be approximated by a linear function.

Efficiency analysis of adding two vectors

In efficiency analysis we assume that the data is distributed - we only consider the time of communication and computation - not the time needed to distribute and gather data.

On each processor we compute

$$z_i = x_i + y_i.$$

The computing time is

$$T(p) = \tau_A \cdot n$$

where n is the length of the array at each processor.

Using the definitions given above, we get the speed-up

$$S(p) = \frac{T(1)}{T(p)} = \frac{N\tau_A}{n\tau_A} = \frac{np\tau_A}{n\tau_A} = p,$$

and the efficiency

$$\eta(p) = S(p)/p = p/p = 1.$$

Hence we conclude that adding two vectors is an <u>optimal</u> parallel process.

Performance model of the inner product

1. Compute
$$a(q) = \sum_{i=1}^{n} x_i y_i$$
on each processor $q = 1, \ldots p$.
2. Send $a(q)$ to all processors and receive on each processor.
3. On each processor, compute
$$a = \sum_{i=1}^{p} a(q).$$

This operation will cost

$$T(1) = (N-1)\tau_A + N\tau_A$$

and

$$\begin{aligned} T(p) &= (n-1)\tau_A + n\tau_A && ((x,y) \text{ locally}) \\ &+ (p-1)\tau_C(1) && (\text{communication}) \\ &+ (p-1)\tau_A && (\sum_{i=1}^{p} a(q)) \end{aligned}$$

Here we assume that

i) $n \geq 10^5$, $N \geq 10^6$

ii) $p \geq 50$

iii) τ_A, τ_C are very small

Thus, we have

$$T(1) \approx 2N\tau_A = 2np\tau_A$$

and

$$\begin{aligned} T(p) &\approx 2n\tau_A + p\tau_C(1) + p\tau_A \\ &= (2n+p)\tau_A + p\tau_C(1). \end{aligned}$$

If we assume that

$$\tau_C(1) = \gamma\tau_A \;\; (\gamma \sim 10 - 5000)\,,$$

we have

$$T(p) \approx (2n+p)\tau_A + p\gamma\tau_A.$$

Recall that $n = N/p$, hence

$$T(p) \approx \left[\frac{2N}{p} + (1+\gamma)p\right]\tau_A$$

A plot of $\frac{2N}{p} + (1 + \gamma)p$ is given below for the three cases:

a) $N = 10^6$, $\gamma = 50$, $\frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^6}{p} + 51p$

b) $N = 10^7$, $\gamma = 50$, $\frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 51p$

c) $N = 10^7$, $\gamma = 1000$, $\frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 1001p$

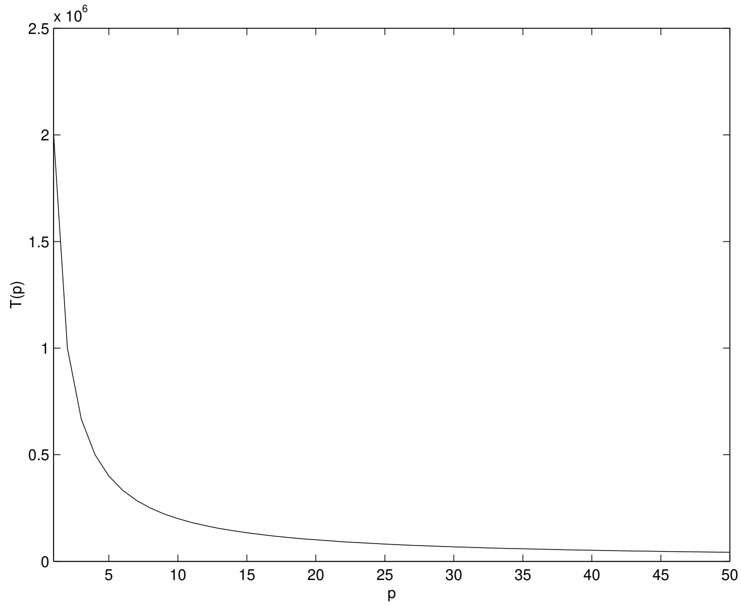


Figure 1: a) $N = 10^6$, $\gamma = 50$

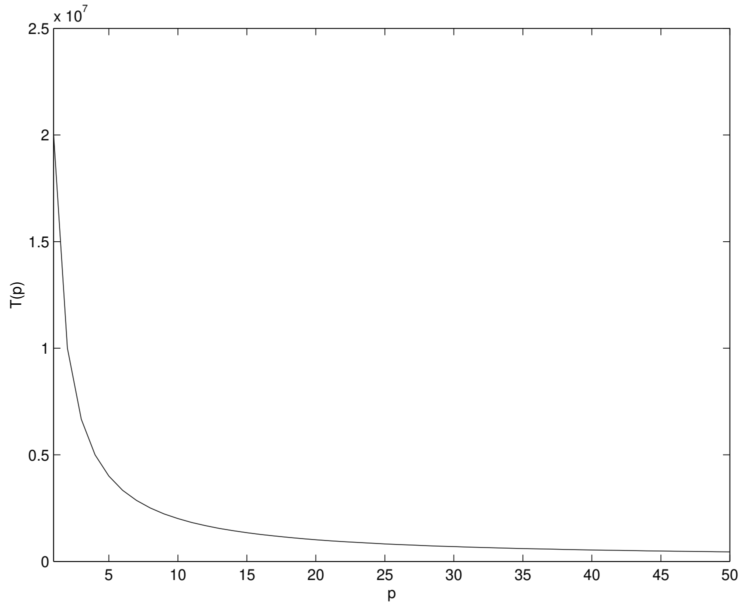


Figure 2: b) $N = 10^7$, $\gamma = 50$

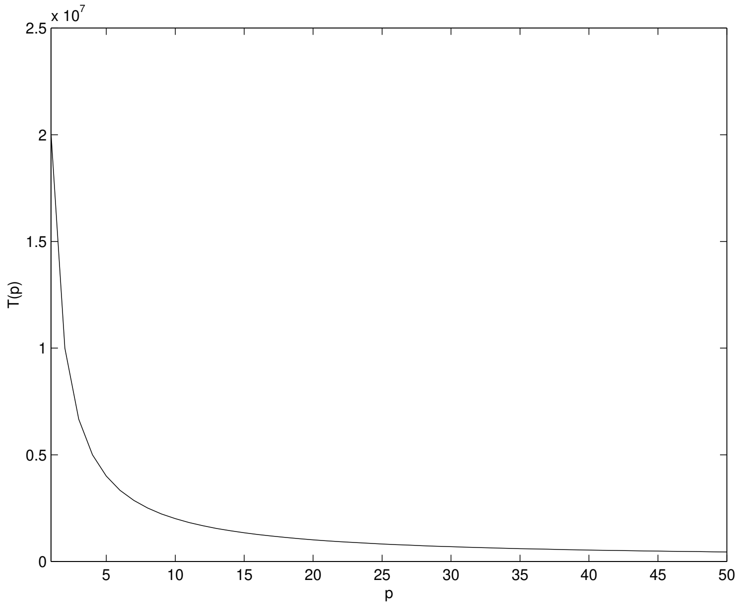


Figure 3: c) $N = 10^7$, $\gamma = 1000$

Speed-up is given by

$$
\begin{aligned}
S(p) = \frac{T(1)}{T(p)} &\approx \frac{2np\tau_A}{(2n + p + \gamma p)\tau_A} \\
&= \frac{2N}{\left(2\frac{N}{p} + (\gamma + 1)p\right)\tau_A} \\
&= \frac{2N}{2N + (\gamma + 1)p^2}p \\
&= \frac{1}{1 + \frac{(\gamma+1)p^2}{2N}}p
\end{aligned}
$$

Since $S(p) \sim p$ is optimal, we must have

$$
\frac{(\gamma + 1)p^2}{2N} \ll 1
$$

in order for this computation to be optimal.

Remarks:

1. Note that
$$\tau_C(1) = \gamma\tau_A$$
where $\gamma \sim 50 - 5000$. An optimal computation requires
$$p \ll \left(\frac{2N}{\gamma+1}\right)^{1/2}.$$
2. Since
$$\frac{T(p)}{\tau_A} = \frac{2N}{p} + (1+\gamma)p,$$
The optimal number of processors p^* can be computed by requiring that
$$\frac{T'(p)}{\tau_A} = 0, \text{ i.e } p^* = \sqrt{\frac{2N}{1+\gamma}}$$
Note that this number is increasing in N and decreasing in γ.
3. For practical purposes, N is usually very large and thus we choose p to be as large as possible. But note that this is not ptimal for small problems.

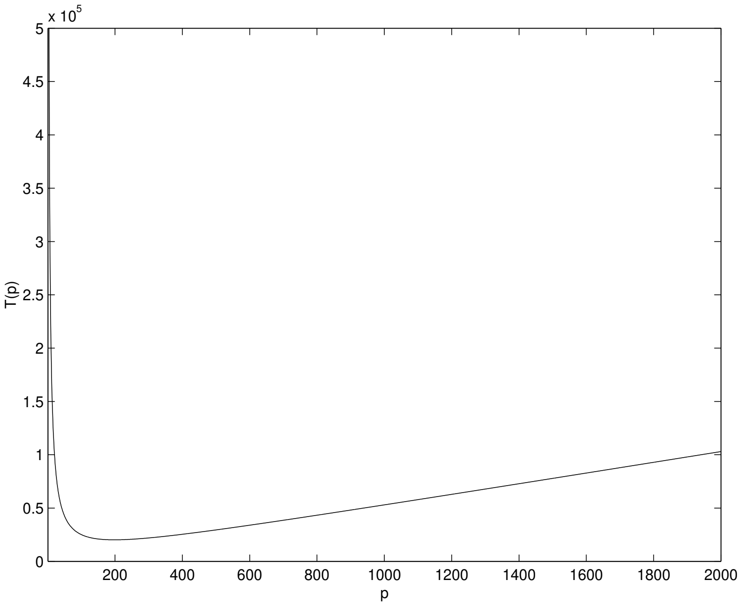


Figure 4: a) $N = 10^6$, $\gamma = 50$, $p^* \approx 198$

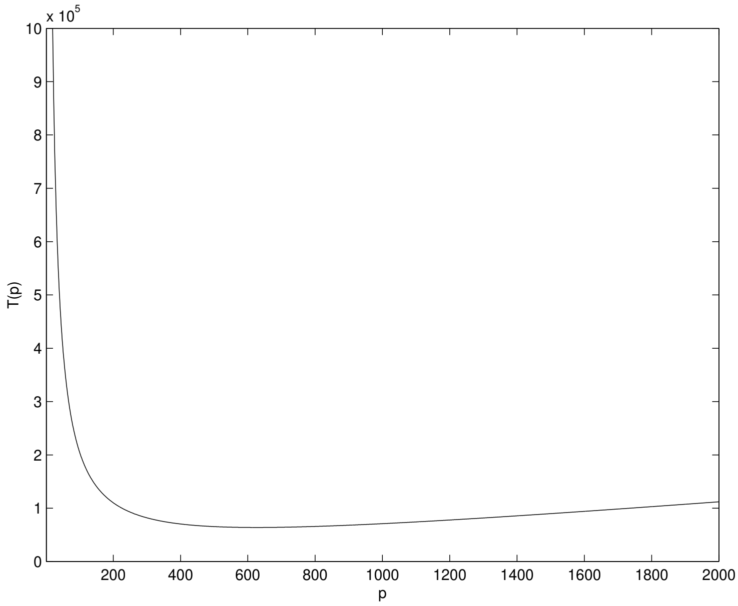


Figure 5: b) $N = 10^7$, $\gamma = 50$, $p^* \approx 626$

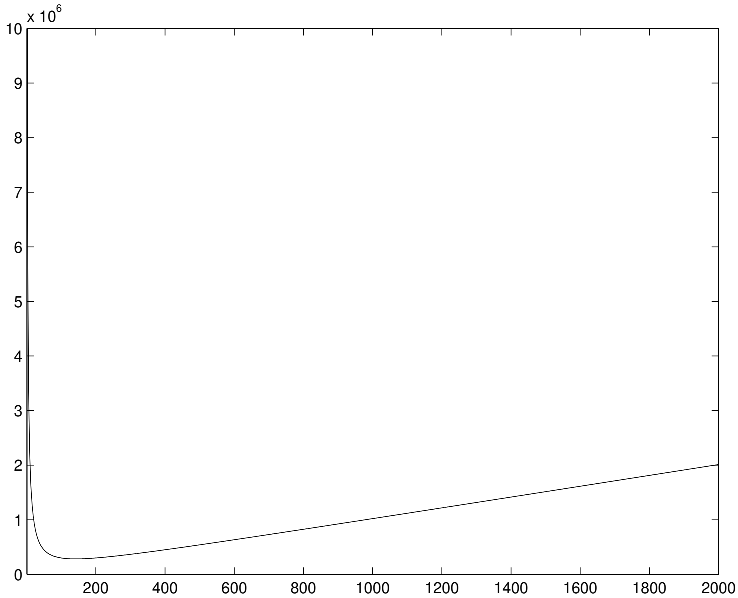


Figure 6: c) $N = 10^7$, $\gamma = 1000$, $p^* \approx 141$

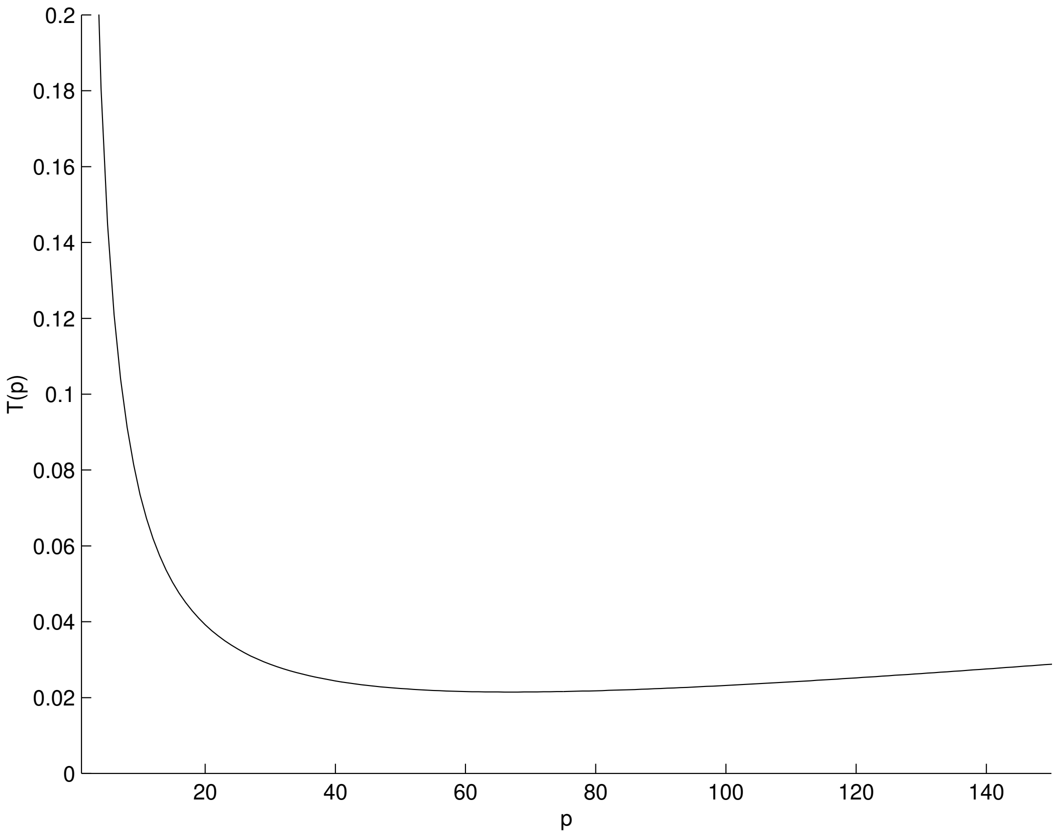
Data for computers installed in Norway

	τ_A	$\tau_C(1)$	$\gamma = \tau_C(1)/\tau_A$
Linux-cluster, Oslo	$3.6 \cdot 10^{-8}$	$1.6 \cdot 10^{-4}$	4444
Origin 2800, Trondheim	$5.5 \cdot 10^{-8}$	$1.7 \cdot 10^{-5}$	309
Origin 2000, Bergen	$8.2 \cdot 10^{-8}$	$9.8 \cdot 10^{-6}$	120
HP SuperDome 9000, Oslo	$2.5 \cdot 10^{-8}$	$2.8 \cdot 10^{-6}$	112

Linux-cluster

$$N = 10^7,\ \gamma = 4444,$$

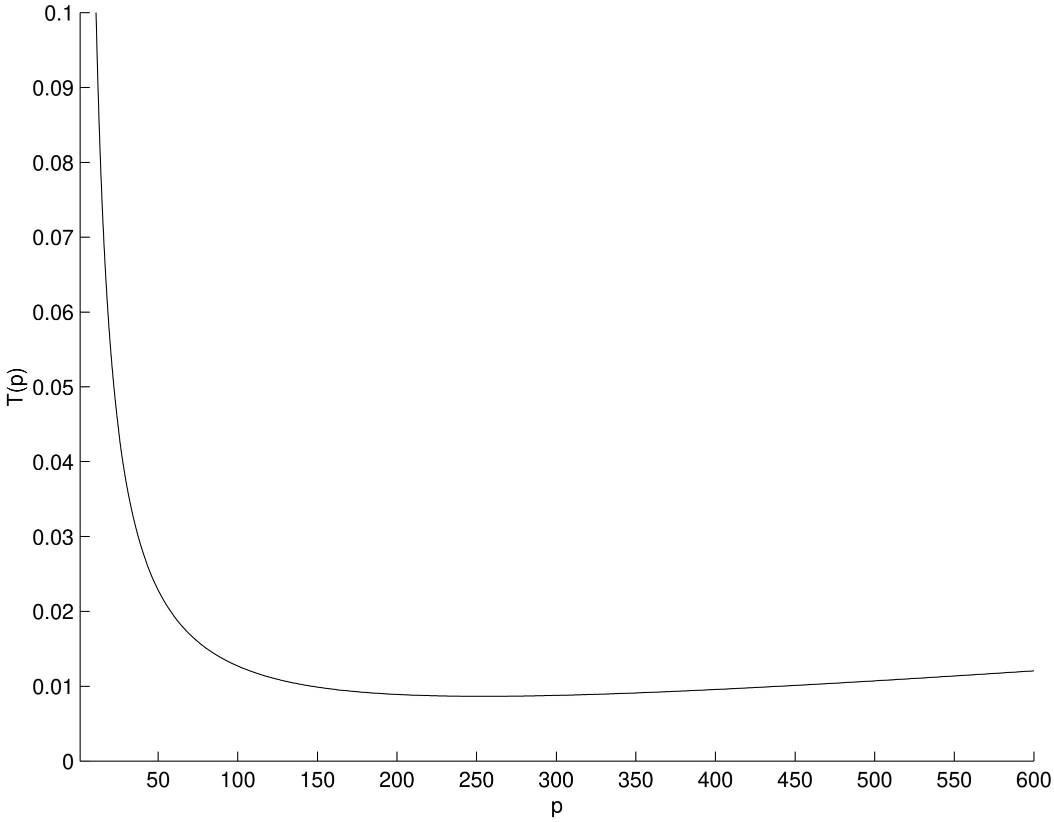
$$p^* = \sqrt{\frac{2 \cdot 10^7}{4445}} \approx 67,\ \frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 4445p$$



Origin 2800

$$N = 10^7,\ \gamma = 309,$$

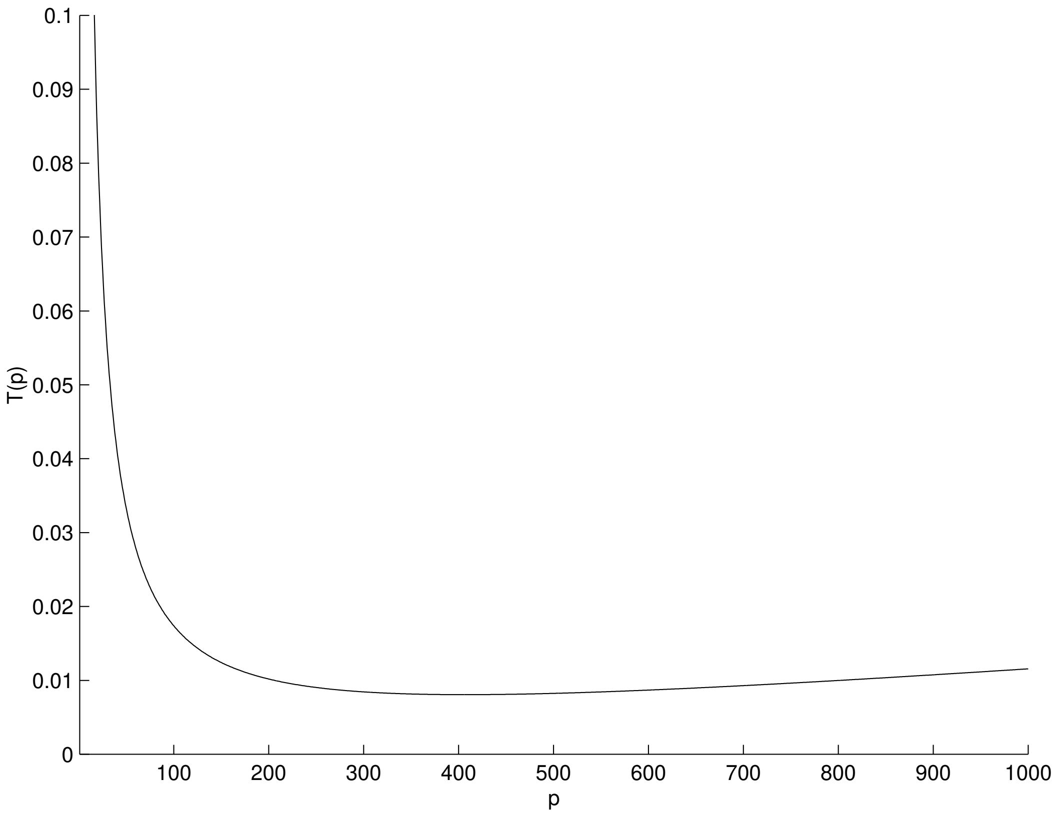
$$p^* = \sqrt{\frac{2 \cdot 10^7}{310}} \approx 254,\ \frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 310p$$



Origin 2000

$$N = 10^7,\ \gamma = 120,$$

$$p^* = \sqrt{\frac{2 \cdot 10^7}{121}} \approx 407,\ \frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 121p$$



HP SuperDome 9000

$$N = 10^7,\ \gamma = 112,$$

$$p^* = \sqrt{\frac{2 \cdot 10^7}{113}} \approx 421,\ \frac{T(p)}{\tau_A} \approx \frac{2 \cdot 10^7}{p} + 113p$$