A Short Course in Parallel Scientific Computing

Technical University of Budapest

Graham Horton, University of Erlangen-Nürnberg

Talk 1 : Introduction to Parallel Scientific Computing

Contents

- Introduction
- Motivation
- Examples
- The parallel computer

Goals, Overview

Goals:

- To introduce parallel scientific computing
- To describe some typical applications
- Demostrate some of the typical problems that have to be solved
- Provide opportunity to gain experience on a parallel computer

Overview:

1. Introduction
2. Fundamentals
3. Partial differential equations
4. Ordinary differential equations
5. Eigenvalues
6. Load balancing

The Group

You:

- Have programming experience in C
- Are interested in parallel scientific computing

Me:

- Graham Horton
- Computer Science Department
  University of Erlangen-Nürnberg
  `graham@informatik.uni-erlangen.de`
- Interests:
  - Parallel Algorithms
  - Load Balancing Methods
  - Stochastic Modeling with Petri-Nets and Markov-Chains

Scientific Computing

Scientific Computing means solving problems with a computer from

- Sciences (esp. Physics, Chemistry)
- Engineering (Automobile, Mechanical, Aerospace)
- Geosciences (Environmental, Oceanography, Weather forecasting)

Problems from these areas often require huge computations

The resources are now becoming available

⇒Scientific Computing is growing in importance

Scientific Computing covers many areas

Most of the problems are described by *differential equations*

These have to be solved on a computer

Introduction - Scientific Computing

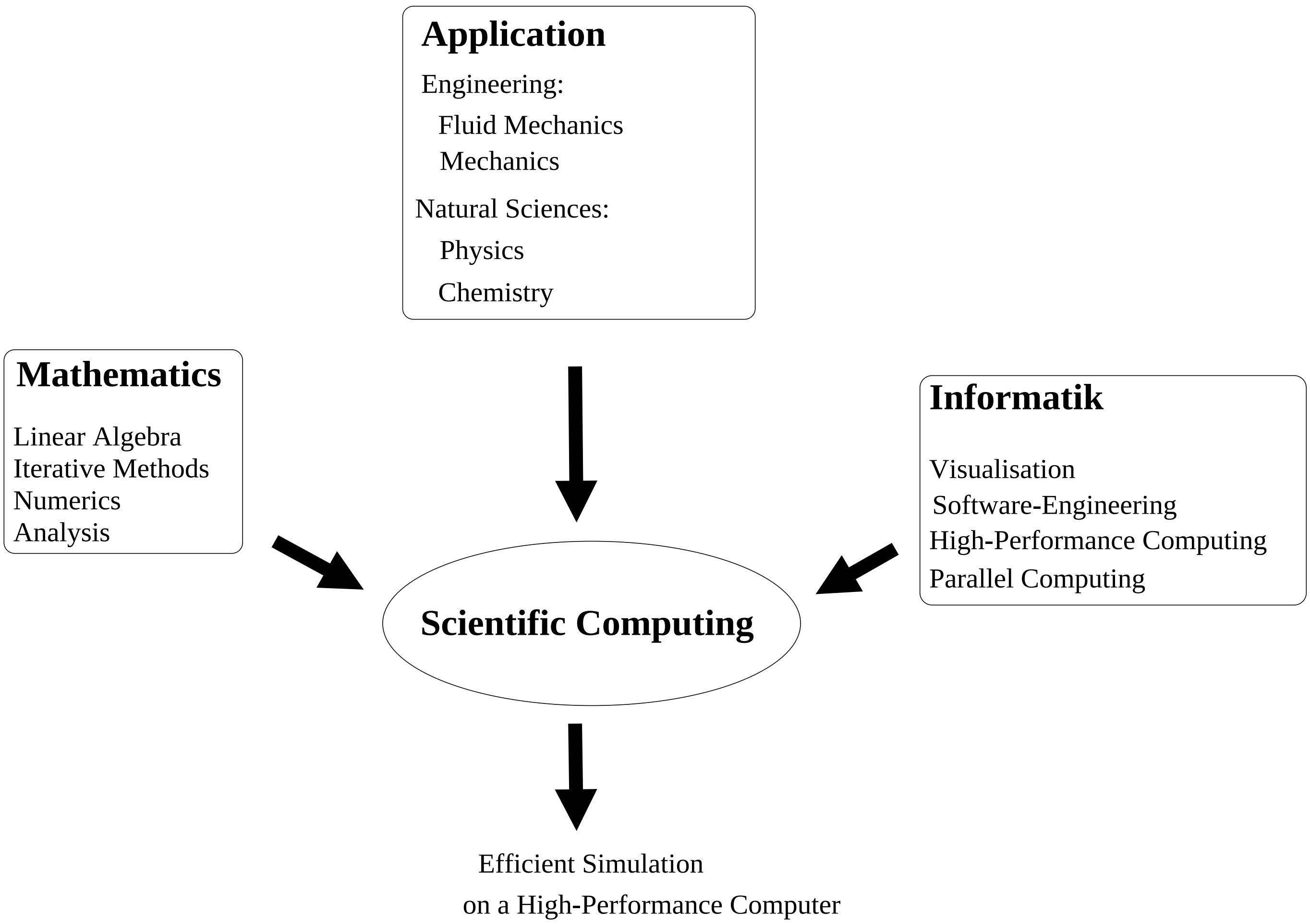
A new field of interdisciplinary research/study, with

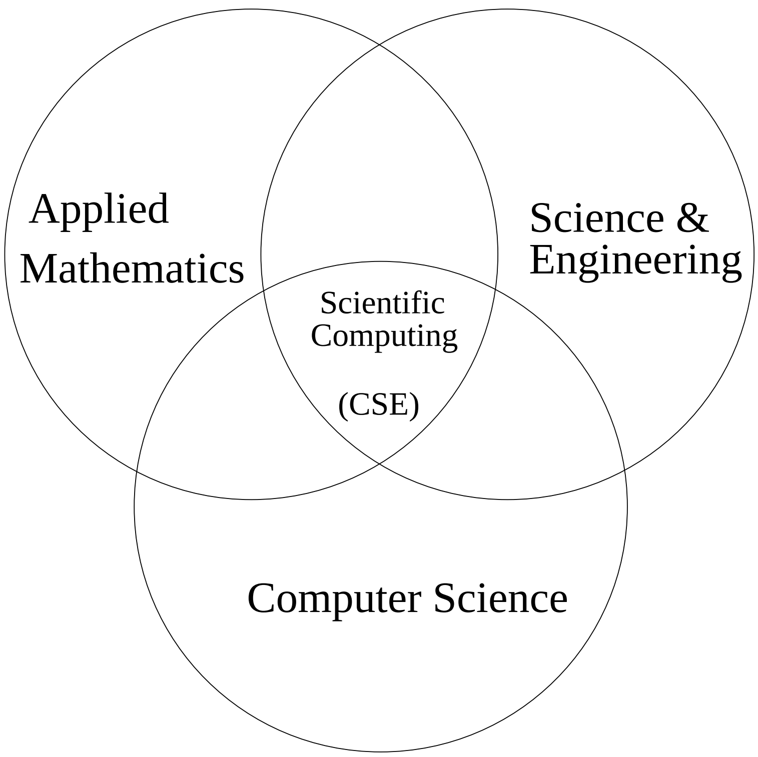
- Mathematics
  - Find an appropriate description of the problem
  - Develop suitable solution algorithms
- Computer Science
  - Parallel programming and computers
  - Visualisation and graphics
  - Software technology
- Individual Scientific and Engineering disciplines
  - Provide problems and applications
  - Experience and knowledge of application field

Scientific Computing is experiencing rapid growth

- USA: Introduction of CSE-courses at many universities
- BRD: Introduction of interdisciplinary research centers
  - IWR - Institut für Hochleistungrechnung (Heidelberg)
  - ICA - Institut für Computeranwendungen (Stuttgart)

Introduction - Scientific Computing



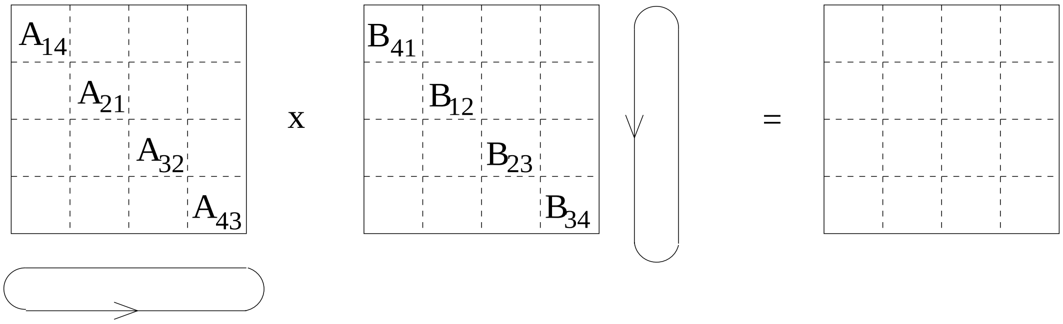


Example: Matrix Multiplication

Multiplication of large, dense matrices $C = A \times B \in I\!R^{n \times n}$

Sequential complexity is n^3, memory complexity is n^2

Parallel method:



Partition matrices and permute them with a block shift

Assign each processor a submatrix a, b, c from A, B, C

```
c = 0
for i = 1 to √p
    c = c + matmul(a,b)
    a = shiftR (a)
    b = shiftD (b)
```

Parallel complexity n^3/p, memory complexity n^2

Complexity at best (with $p = n^2$) equal to n

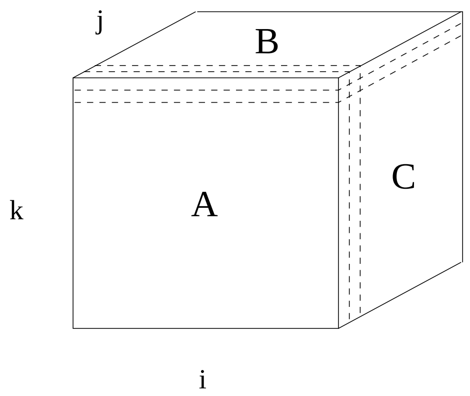
Trivial parallelism and load balancing

But can we do better?

Matrix Multiplication

We can achieve a much faster parallel algorithm

Use $p = n^3$ processors with cube topology



Distribute A to all i, k-planes

Distribute B to all k, j-planes

Each processor computes $t_{ijk} = a_{ij} * b_{jk}$

Subsequently summation $c_{ij} = \sum_k t_{ijk}$

Parallel complexity: $\mathcal{O}(\log n)$, memory n^3

This example reveals several issues:

- Conflict between computational and memory complexity
- Granularity / degree of parallelism
- Topological requirements (Hypercube)

Status of Parallel Computing

We are at present in the third phase of parallel computing

- The theoretical - speculative phase ($\ldots 1983$)
  - Fundamental and theoretical considerations (Amdahl, Lee, Minsky, Flynn)
  - Development of some important parallel methods (Batchers bitonic sort, parallel Jacobi eigenvalue method)
  - Hardly any parallel computers available
- The euphoric phase ($1983 \ldots 1994$)
  - First machines become available (also commercially)
  - Immense support for the hardware industry by US defence programs (In Europe: Transputer, SUPRENUM)
  - Many applications start to be solved by parallel processing
  - Growth in number of conferences, journals etc.
- The "sobering up" phase (since 1994)
  - Natural reaction after euphoric phase
  - Dwindling government support (end of the cold war)
  - Disappearance of several key manufacturers (reasons?)
  - Confrontation with difficulties (hardware stability, programming, debugging, portability)

Status of Parallel Computing

"... The parallel computing environment is still emerging from the 1960s. Users still struggle to understand (and overcome) the memory-management and message-passing strategies devised by the vendors. . . ."
D. P. O'Leary [1]

"Asked to complete the sentence "Parallel processing is . . .", an honest assessment must necessarily include the words "painful" and "inevitable"."
H. S. Morse [2]

"If you aren't *really sure* you need massively parallel computing, you had probably better think twice before getting involved with it."
H. S. Morse [2]

"[Our parallel] turbulence calculation . . . broke new ground. . . We have . . . first glimpses of Kolmogorov inertial range turbulence."
P. R. Woodward [6]

Morse's "Pain" and "Inevitability"

"Pain":

- Difficulty and costs of parallel programming
- Instability of HW and SW
- Transience of manufacturers
- Lack of standards and portability

"Inevitability" (technological arguments):

- Clock speed limits
- Memory bandwidth
- Packaging, cooling, power supply
- Maintenance: specialisation and cost

Motivation for Parallel Computers

What is a parallel computer ?

Several computational units connected together

All work on the same problem

The computational units can be :

- Components of an ALU (vector processor)
- Just ALUs (array processor)
- Complete processors (multiprocessors)
- Complete computers (multi-computer)

Ideas:

- Several units can be faster than one
- Many small units are cheaper than one large one
- Achieve same computational power with cheaper technology
- Reduce technological difficulties
- Scalability, expandability

Problems of Parallel Computing

Which (new) problems occur in parallel processing?

- *Parallelism of the algorithm.*
  - Can the existing sequential algorithm be parallelised?
  - Should it be modified / must it be replaced?
  - How efficient is it?
- *Load balancing.*
  - All processors should have equal load
  - Idle times must be avoided
  - Dynamic or static?
- *Minimisation of communication overhead.*
  - Mapping subproblem→processor influences communication times
  - → load balancing
  - Mapping problem
- *Implementation / Portability.*
  - Porting programs with reasonable cost ?
  - Must the application be completely re-programmed?
  - Is the parallel program portable between parallel computers?
  - Are new data structures necessary?

Parallelism

The only way to substantially increase computational power is *parallelism*

The most compute-intensive applications are almost always *simulations*

The phenomena to be simulated are almost always themselves parallel!

The sequentiality of programming languages seems artificial in these cases

Example : Simulation of the solar system

`pos[i]` is the position of the i-th planet, `vel[i]` its velocity

Classical processing :

```
for i = 1 to n
    pos[i] = pos[i] + Δt * vel[i];
```

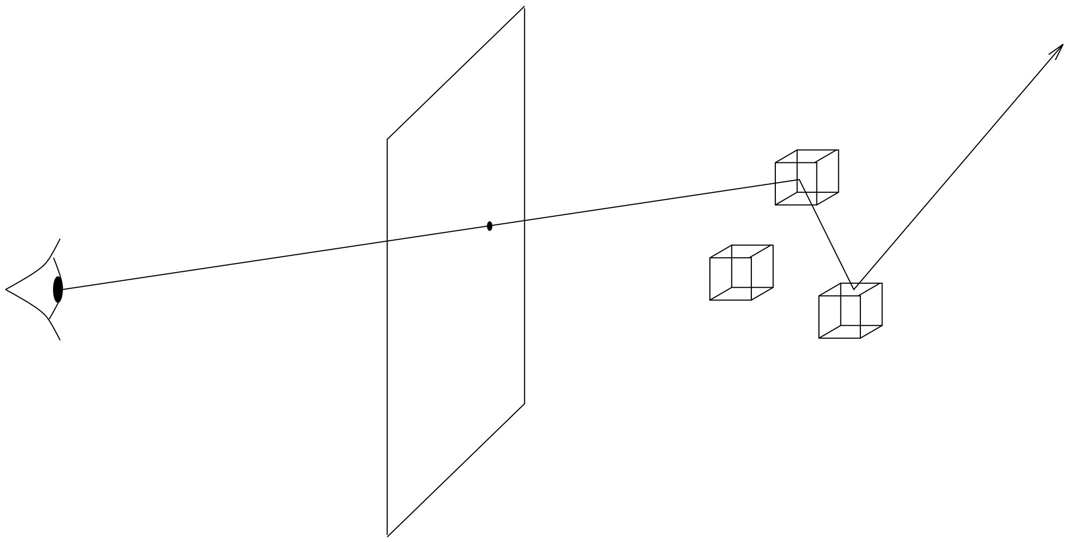
What we would really like to be allowed to write is :

```
∀ i ∈ {1:n}
  pos[i] = pos[i] + Δt * vel[i];
```

This clearly reveals the possibility for simultaneous computation

Example Application - Visualisation

Ray tracing is a technique for realistically representing scenes



Follow a light ray backwards from the eye into the scene

Parallel computation of each pixel, as each is independent of the others

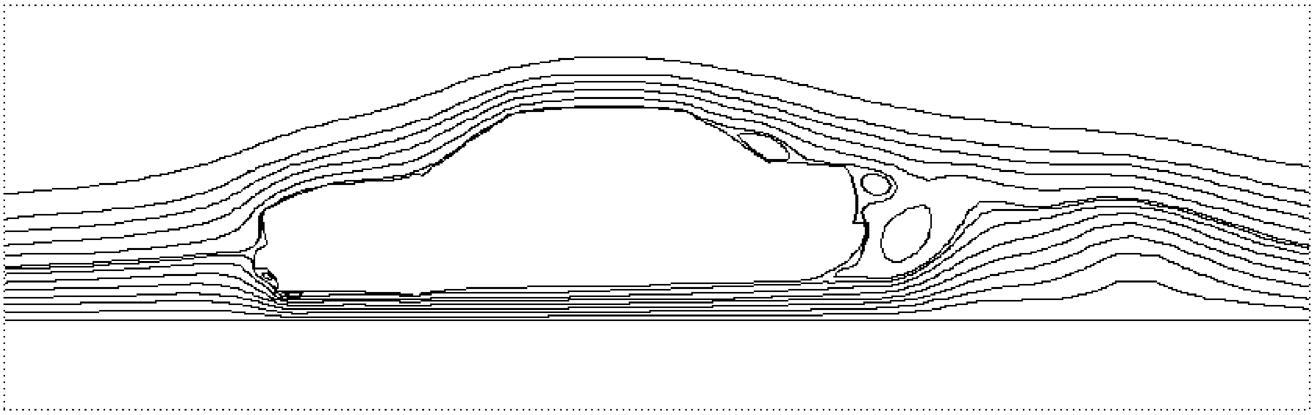
High-quality images have $\geq 4000 \times 4000$ pixels

Animations require 25 pictures per second

A one-hour film requires the computation of $\approx 1.4 \times 10^{12}$ pixels

Example Application - Fluid Mechanics (CFD)

Computation of the flow around a vehicle



We are interested in :

- The c_w value (air resistance)
- Combustion (fuel efficiency)
- Ventilation (temperature and fresh air distribution)

Example Application - Fluid Mechanics (CFD)

Solution of the Navier-Stokes equations

$$\begin{aligned}
\frac{\partial(\rho U)}{\partial x} + \frac{\partial(\rho V)}{\partial y} &= 0 \\
\frac{\partial}{\partial x}(\rho U^2 - \mu\frac{\partial U}{\partial x}) + \frac{\partial}{\partial y}(\rho UV - \mu\frac{\partial U}{\partial y}) &= -\frac{\partial P}{\partial x} \\
\frac{\partial}{\partial x}(\rho UV - \mu\frac{\partial V}{\partial x}) + \frac{\partial}{\partial y}(\rho V^2 - \mu\frac{\partial V}{\partial y}) &= -\frac{\partial P}{\partial y}
\end{aligned}$$

CFD is one of the most compute-intensive application areas

- Very fine grids ($\geq 500 \times 500$ $(\times 500)$) points
- Many floating point operations per point (≥ 3000)
- Many equations (temperature, reactions, etc.)
- Many time-steps

The computation of the Space-Shuttle would last $>$ 1 year on a Cray!

CFD applications in

- Automotive and Aerospace Engineering
- Medicine
- Chemical Industry

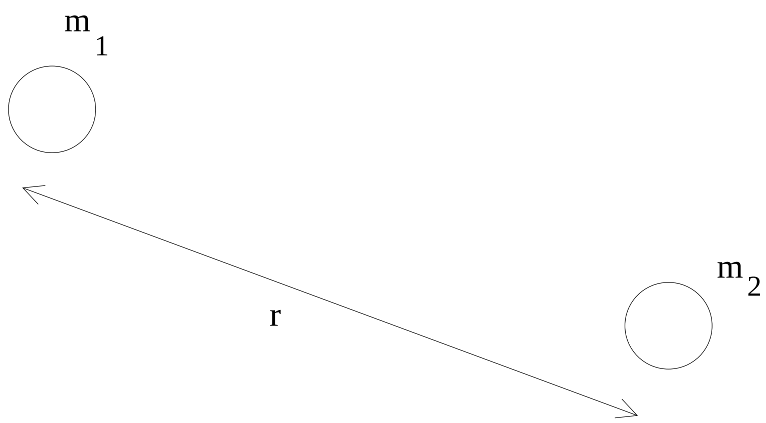
Example Application - Astronomy

Simulation of galaxies

Formation of spirals, clusters, etc.

Solution of gravitation laws (Newton) for many ($> 10^6$) bodies

$$F_{1,2} = \Gamma \frac{m_1 m_2}{r^2}$$



With n bodies $O(n^2)$ operations per time-step

Long intervals and large number of bodies yield large computation times

Solution of a large system of ODEs

$$\frac{d\vec{u}}{dt} = f(\vec{u})$$

Also particle physics : deduce macro-states from particle laws

Example : Pressure is sum of the impulse of impacting molecules

Types of Parallelism

There are three main types of parallelism :

- Divide and Conquer.
  - Often related to recursion
  - In many Computer Science algorithms
  - Example : Maximum of n numbers :

$$\max_{1\leq i\leq n} a_i = \max(\max_{1\leq i<n/2} a_i,\ \max_{n/2\leq i\leq n/2} a_i)$$

  - Example : Sorting of n numbers :

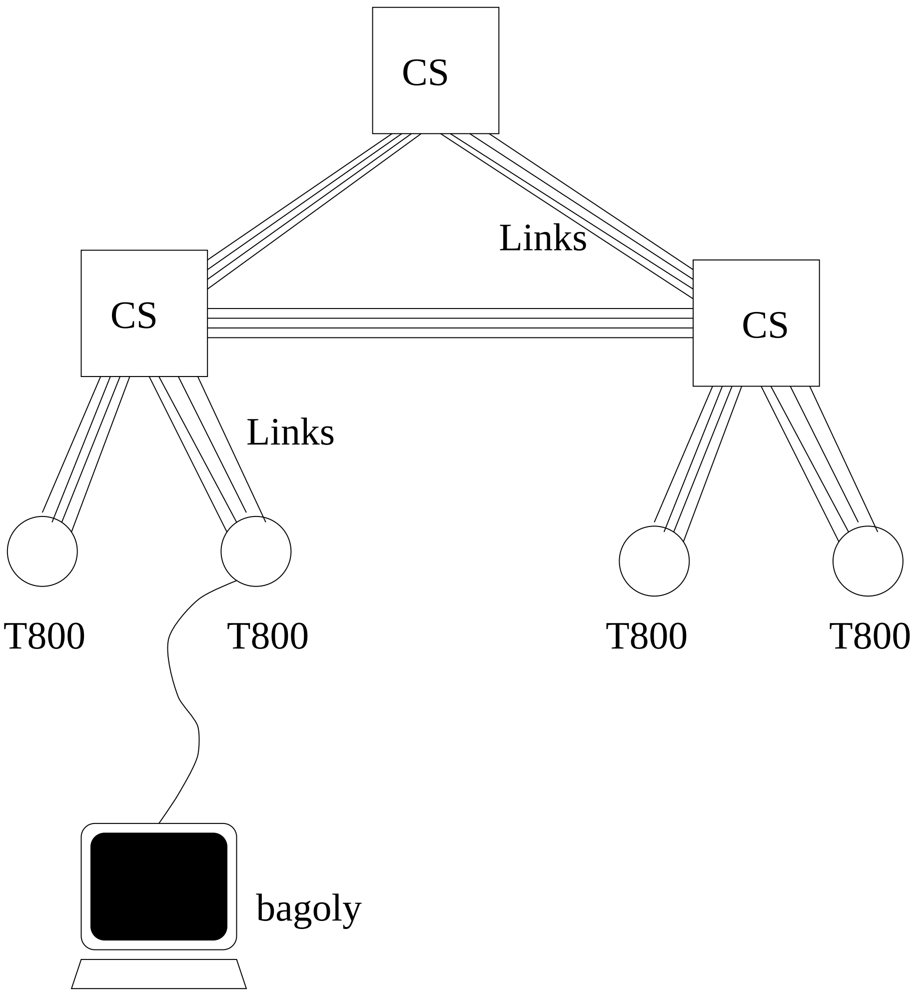
$$sort_{1\leq i\leq n} a_i = merge(sort_{1\leq i<n/2} a_i,\ sort_{n/2\leq i\leq n/2} a_i)$$

- Parameter parallelism
  - A problem must be solved for different parameter values
  - Trivial parallelisation
  - High efficiencies can be achieved
  - Example : Which petrol/air ratio gives most efficient combustion?
- Data parallelism
  - Large volume of data is distributed across processors
  - Most important type for scientific computing applications
  - Example : Matrix-vector multiplication

Programming the Meiko Computing Surface

We will use the Meiko Computing Surface for experiments

- It is connected to the computer `bagoly.mmt.bme.hu`
- Has 64 T800 Transputers
- Each T800 has 4 links to other Transputers
- Links are conencted via crossbar switches
- The crossbar switches are programmable
- Yields various topologies (tree, grid, ring, etc.)



Programming the Meiko Computing Surface

You will need the path in your environment

`/usr/bin/` (should be there anyway)

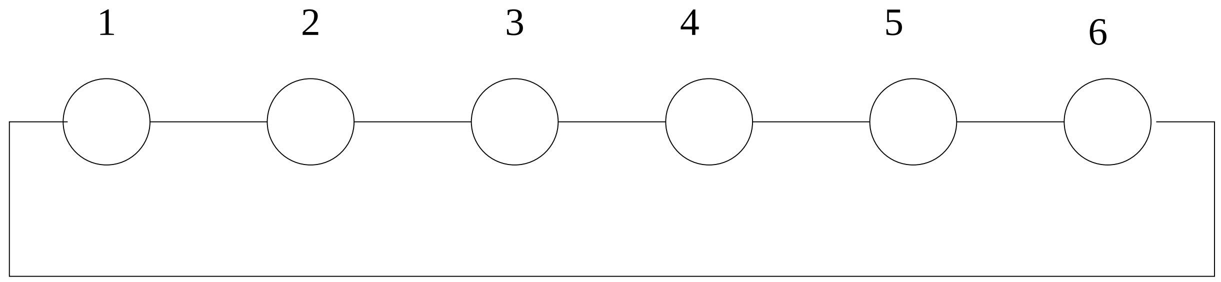
Programs for the Meiko are written in C and compiled with the command

```
mcc -o <fn> <fn.c> -lm -lrcs -lcs -lcsn
```

I have written a configuration program `loader`

`loader` configures the transputers as a ring

Example: `loader <fn> 6`



The command `tps -m` shows the activity on the parallel computer

Experiment with the parallel "Hello world" program for use of software

Programming the Meiko Computing Surface

All programs can be found under ....

We will look at three problems

1. Parallel integration (approximation of π)
2. Parallel solution of an elliptic PDE (heat equation)
3. Parallel solution of a system of ODEs (planetary motion)

Parallel “Hello World” Program

```
#include <stdio.h>
#include <math.h>
#include <csn/csn.h>
#include <transputer.h>
#include <cstools/cstools.h>


int     procno;        /* Number of this processor          */
int     nprocs;        /* Number of processors              */
InChan  fromleft;      /* Communication channel from left   */
OutChan toright;       /* Communication channel to right    */
InChan  toleft;        /* Communication channel to left     */
OutChan fromright;     /* Communication channel from right  */


void init()              /* Set values of important variables */
{                        /* Always call this first!           */
  int argc;
  char **argv;

  csn_init(&argc,&argv);
  cs_import("int procno",&procno);
  cs_import("int nprocs",&nprocs);
  cs_import("InChan fromleft",&fromleft);
  cs_import("OutChan toright",&toright);
  cs_import("InChan fromright",&fromright);
  cs_import("OutChan toleft",&toleft);
}
```

Parallel "Hello World" Program

```
void sync()                  /* Synchronisation of all processors  */
{                            /* by forwarding a message */
  int i=0;

  if (nprocs > 1)
    if (procno == 1)
    {
      cwrite(toright,&i,sizeof(int));
      cread(fromleft,&i,sizeof(int));
    }
    else
    {
      cread(fromleft,&i,sizeof(int));
      cwrite(toright,&i,sizeof(int));
    }
}

void main()
{
  int time1, time2;

  init();
  sync();
  sync();
  time1 = ticks();
  sync();
  time2 = ticks();
  printf("Hello world! Here is processor %d of %d \n",procno,nprocs);
  printf("Proc. %d : Synch. time : %lf Sec. \n\n",procno,(time2-time1)*6.4E
}
```