Parallel Algorithms for Data Mining

Peter Christen
Computer Sciences Laboratory, RSISE
Australian National University

peter.christen@anu.edu.au
http://csl.anu.edu.au/∼peter

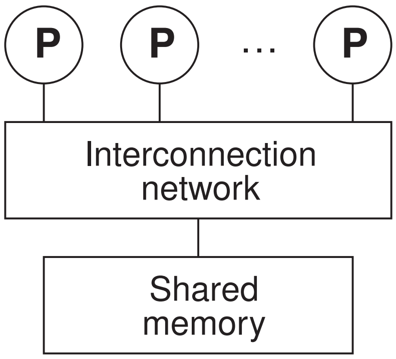
17 May 2000

Funded by grants from the *Swiss National Science Foundation* and *Novartis Stiftung*

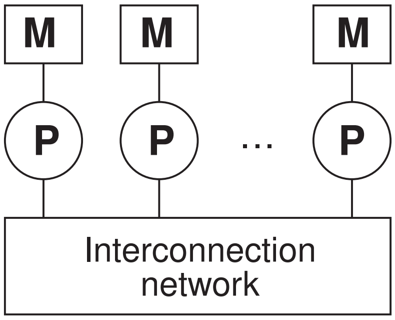
Why Parallel Computing?

- **Large data sets and long processing times**
  (e.g. simulations in physics and chemistry, weather forecast, etc.)
- **Limitations of sequential computers**
  Processor speed, I/O- and memory bandwidth
- Many applications and algorithms contain parallelism
  (e.g. pipelining, domain decomposition)
- **Data Mining**: Data sets from Giga-Bytes to Peta-Bytes, several scans over data set needed, complex algorithms

Parallel Architectures



Shared memory



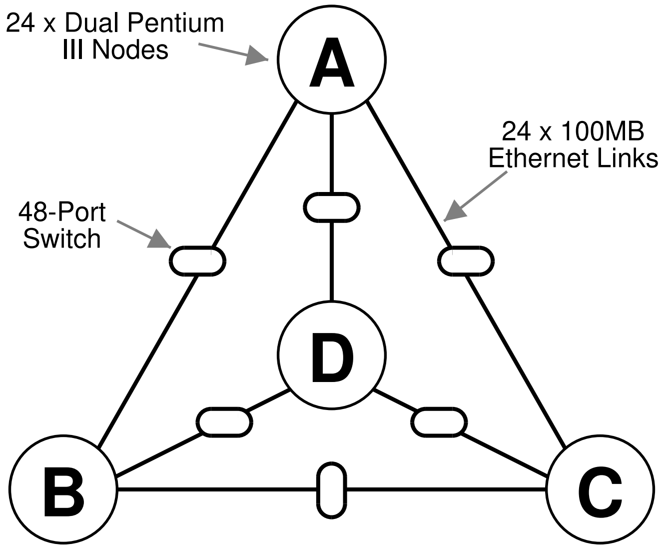
Distributed memory

Parallel Architectures: Two Examples

CSIRO Capricorn	**ANU DCS Bunyip**
– Sun Enterprise 4500 Server	– Beowulf Linux Cluster
– Shared memory (SMP)	– Distributed memory
– 12 UltraSPARC 400 MHz RISC (8 Mega-Bytes cache each)	– 96 Dual Pentium III 550 MHz
– 6,912 Mega-Bytes main memory	– 36,864 Mega-Bytes main memory (384 Mega-Bytes per node)
– 250 Giga-Bytes disk storage array (RAID)	- 1,305.6 Giga-Bytes disk (13.6 Giga-Bytes per node)
	– 100 Mega-Bit Ethernet network

According to the last `www.top500.org` (Nov'99) the ANU Beowulf is Australia's fastest supercomputer.

Parallel Architectures: ANU Beowulf Topology



Parallel Architectures: ANU Beowulf Implementation

Different Kinds of Parallelism

- **Functional Parallelism:** Each processor runs a sub-job, the result is passed to the next processor (pipeline principle).
- **Data Parallelism:** All processors do the same job on different parts of the data (domain decomposition).

- **Master-Worker:** Master process distributes tasks to worker processes which return result back. Good if workers can operate independently.
- **Single-Program Multiple-Data (SPMD):** The same program runs on all processors, but on different sub-sets of the data.

Parallel Programming

- **Message Passing:** Calls to communication routines, e.g.
  SEND(data, P2) or *BROADCAST(vector, P0)*
  Mainly on distributed memory architectures → PVM, MPI
- **Threads:** Program parts that can run independently.
  Mainly on shared memory architectures → OpenMP, Pthreads
- **Parallel Compilers:** Extensions of languages with parallel statements, e.g.
  DISTRIBUTE A (BLOCK) ONTO P
  DO IN PARALLEL ...
  → High Performance Fortran (HPF)

Parallel Performance

- **Speedup:** Sequential time divided by parallel time: $Sp(p) = T_1/T_p$
  Desired: $Sp(p) = p$ (hard to achieve).

- **Efficiency:** Speedup divided by the number of processes: $Ef(p) = \frac{Sp(p)}{p} \leq 1$

  Sometimes: Super-linear speedup $Sp(p) > p \rightarrow Ef(p) > 1$
  (Cache, memory and I/O effects, etc.)

- **Scalability:** Efficiency often drops as the number of processes is increased. Scalability gives a measure how much the data size has to be increased to get the same efficiency on more processes.

Parallel Obstacles

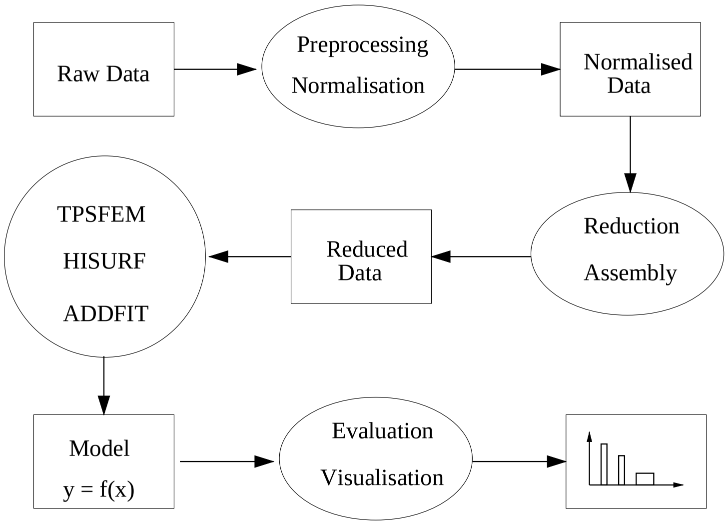
- **Amdahl's Law:** Most algorithms and programs contain sequential parts, which limit maximal speedup and inhibit scalability, e.g. 10% sequential code $\rightarrow$ Maximal speedup 10!

- Balancing the load (distributing work onto processors) can be hard to achieve.

- Data distribution can become a bottleneck (e.g. if all processors are connected to only one I/O system).

- Parallel programs often have to be adapted to a given architecture to get maximum performance.

Data Mining Cycle



Example: Assembly

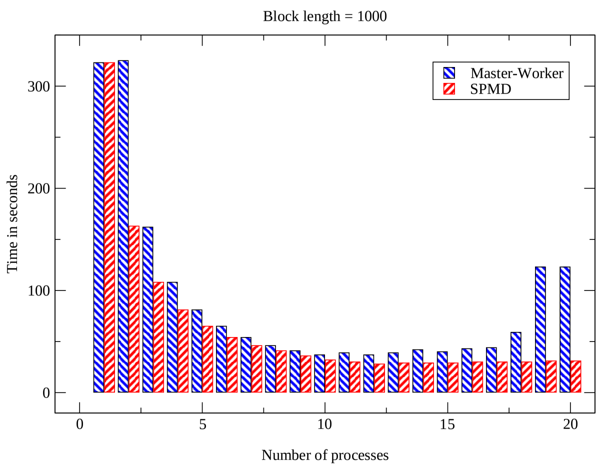
- Each data record adds some values into a matrix.
- The whole data set has to be read only once.
- The size of the matrix is independent of the number of data records.
- Parallelism is easy to achieve: Each processor only reads a part of the data set and adds into a local matrix.
- Reading and assembling is done in blocks of a given size.

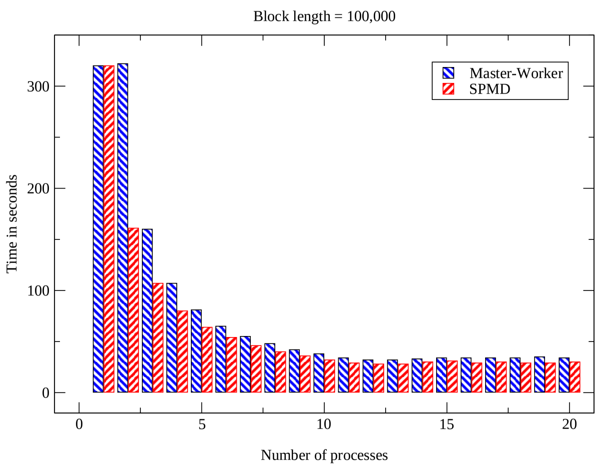
Parallel Assembly - Two Implementations

Master-Worker	SPMD
– Master process controls assembly – Master sends messages to workers with start position in file and number of records to assemble – After assembling a block, worker sends $ready$ message back to master and gets next task	– All processes compute distribution – Each process reads and assembles n/p data records – No communication needed during assembly

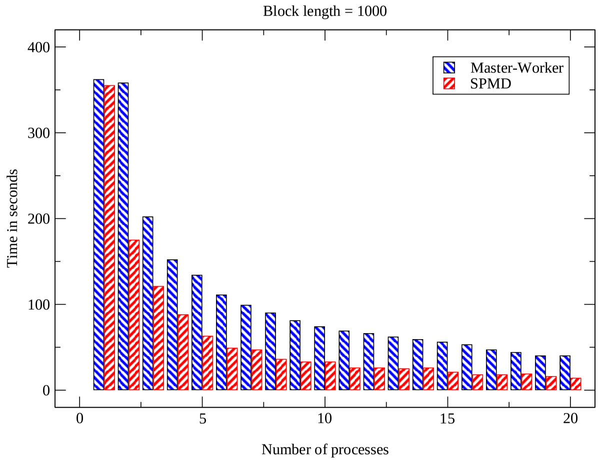
After the assembly is finished, the local matrices are collected and summed on the host-processor (Reduce operation).

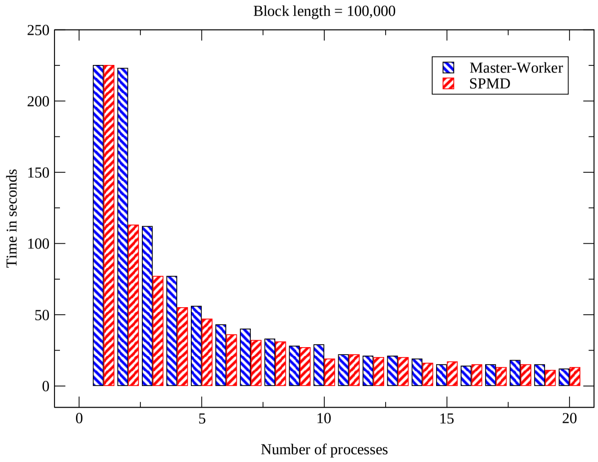
Assembly on Sun Enterprise



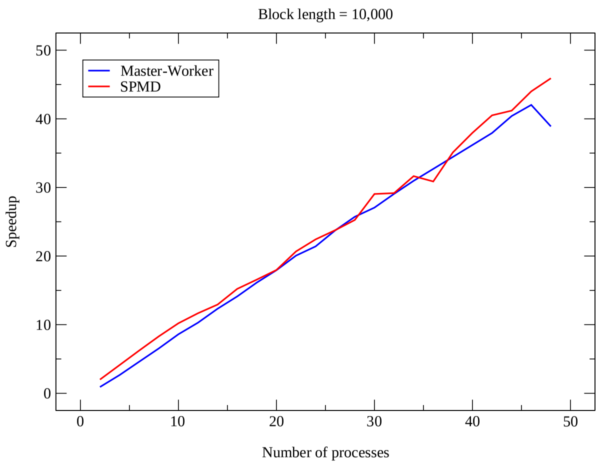


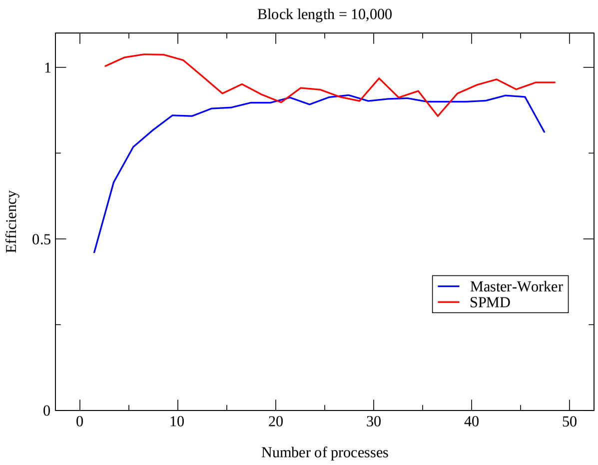
Assembly on Beowulf





Assembly on Beowulf: Speedup and Efficiency





Outlook

- Parallel computing can help to solve bigger and more complex problems.
- It can speed up existing applications.
- Not all applications parallelise well or yield in good speedup and scalability.
- Good parallel programs should be scalable both in data size (number of records) and number of processors.
- Parallel programming is still complicated and cumbersome (run-time effects).