ERIM Open Lecture

Introduction to Parallel Computing

Tommi Tervonen

Econometric Institute, Erasmus School of Economics

Motivation: why parallelize?

- Estimate complex models faster (hours instead of months)
- Run computational experiments with more instances
- Perform exploratory analyses on large data sets

LISA: the Dutch National Cluster

- Cores: 6528 (8 or 12-core nodes)
- RAM: 16TB
- Peak performance: 46 Tflop/s ($4.6 * 10^{13}$)
- Disk: 100TB for the /home
- OS: Debian Linux AMD64
- Bandwidth: 1600MB/sec between nodes

Contents

1 Processes and threads

2 Inter-process communication

3 Synchronization

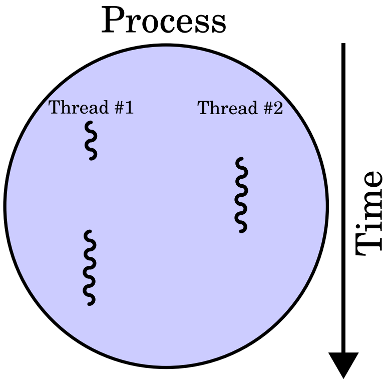
4 Parallel computing architectures

5 GPU computing

6 Examples

Processes and threads

- Kernel processes are the base unit of scheduling
- Processes can contain multiple threads
- Threads within the same process share the address space
- Threads context switch a lot faster than processes
- Inter-process communication



Problems with sharing an address space

```
void loop_over_values(int *counter, int limit) {
  assert(*counter <= limit);

  while (*counter != limit) {
    printf("%d\n", *counter);
    (*counter)++;
  }
}
```

Inter-process communication

- File
- Signal (kill -9 pno)
- Socket (stream)
- Pipe (cat file | grep tommi)
- Message queue (like pipe but with packets)
- Named pipe (file behaving as pipe)
- Shared memory
- Memory mapped file
- Message passing (no sharing)

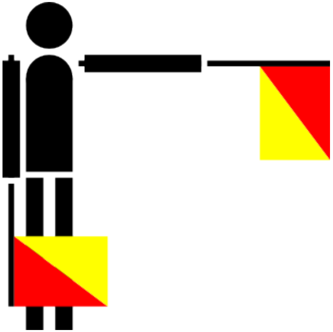
Synchronization: mutual exclusion

- Mutex is a syncronization primitive that allows atomic operations
- `test-and-set(mutex)`
- `unset(mutex)`

```
void loop_over_values(int *counter, int limit,
      mutex* m) {
   while(test-and-set(m) == 0) { }

   assert(*counter <= limit);

   while (*counter != limit) {
      printf("%d\n", *counter);
      (*counter)++;
   }
   unset(m);
}
```

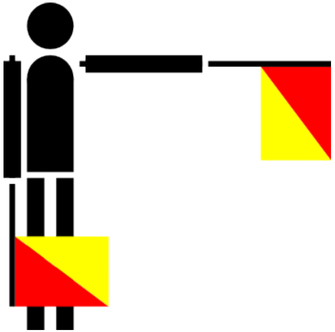
Semaphore

- P (proberen) to obtain one resource (wait until one becomes available)
- V (verhogen) to release one resource (possibly signal a waiting process to restart)
- Semaphores include a queue for the waiting processes (mostly FIFO)



Semaphore

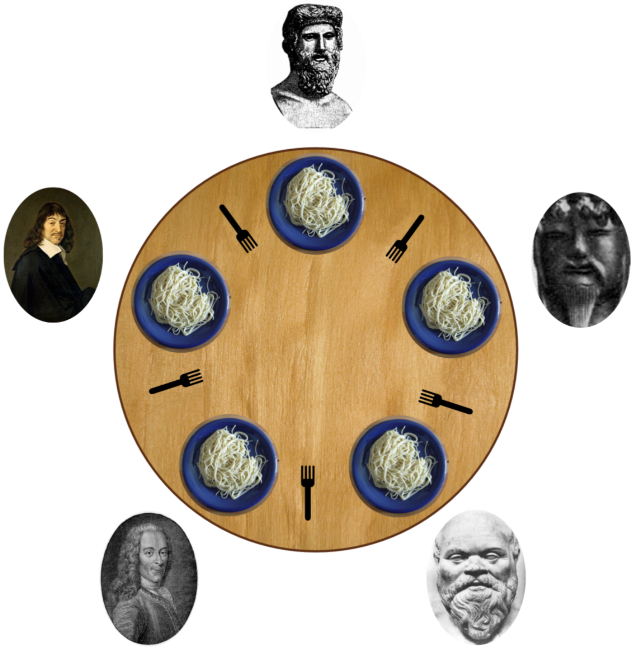
- P (proberen) to obtain one resource (wait until one becomes available)
- V (verhogen) to release one resource (possibly signal a waiting process to restart)
- Semaphores include a queue for the waiting processes (mostly FIFO)
- *Counting semaphores* allow to track multiple resources
- *Binary semaphores* are mutexes with process suspension instead of busy waiting



Critical section

- Critical sections are pieces of code that access a shared resource (data in our case)
- Must always be terminated (simple computation)
- Below, the critical section computation can take long to terminate

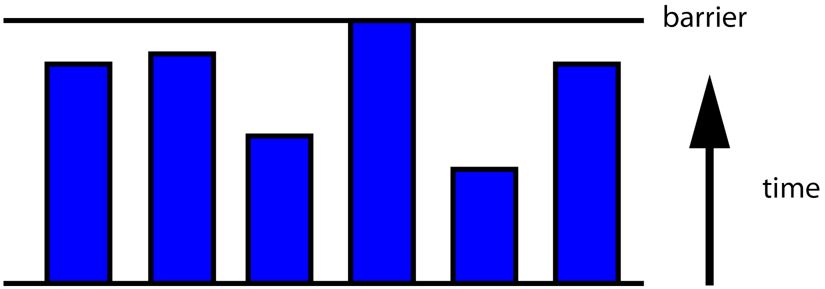
```
void loop_over_values(int *counter, int limit,
    semaphore* s) {
  assert(*counter <= limit);

  semaphore_P(s);
  while (*counter != limit) {
    printf("%d\n", counter);
    (*counter)++;
  }
  semaphore_V(s);
}
```



Monitor

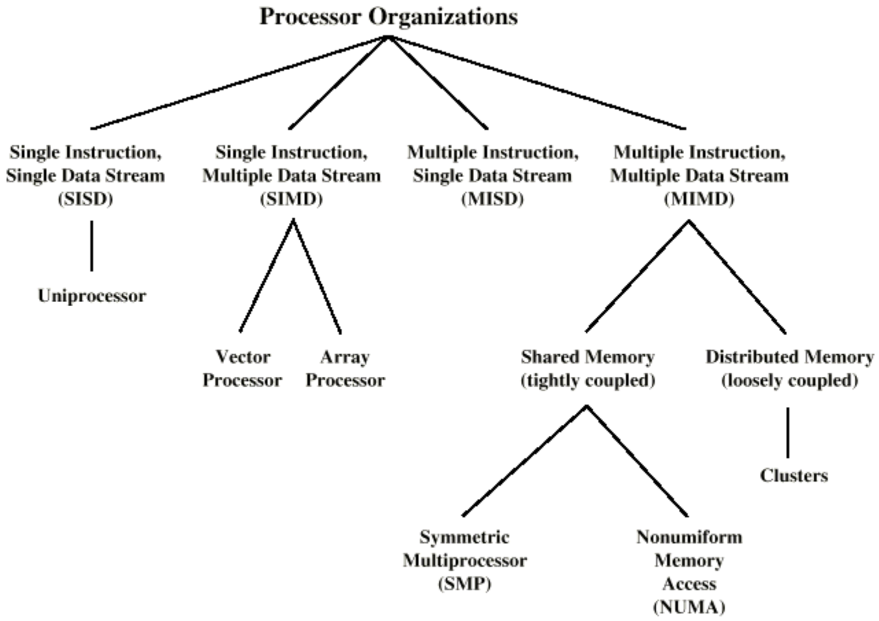
- Synchronizes access with a lock object *and* a condition variable
- Allow threads to give up exclusive access to a resource and wait for a condition to be met
- In Java (`synchronized` keyword): thread-safe methods (only 1 thread at a time may occupy the object)

Barrier

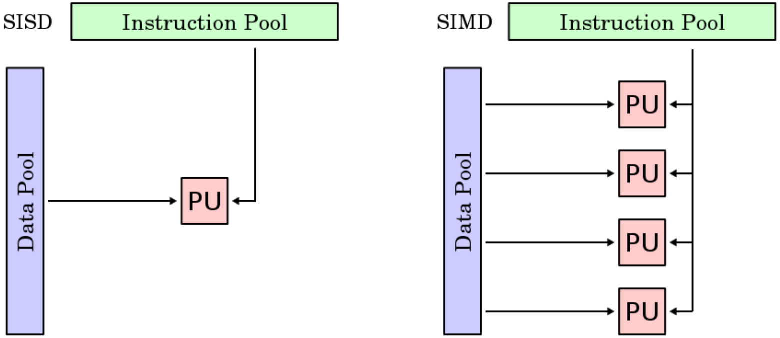


- Barrier synchronizes multiple processes to block until all have reached the barrier

Parallel computing architectures


Processor Organizations
Single Instruction, Single Data Stream (SISD)
Single Instruction, Multiple Data Stream (SIMD)
Multiple Instruction, Single Data Stream (MISD)
Multiple Instruction, Multiple Data Stream (MIMD)
Uniprocessor
Vector Processor
Array Processor
Shared Memory (tightly coupled)
Distributed Memory (loosely coupled)
Clusters
Symmetric Multiprocessor (SMP)
Nonumiform Memory Access (NUMA)

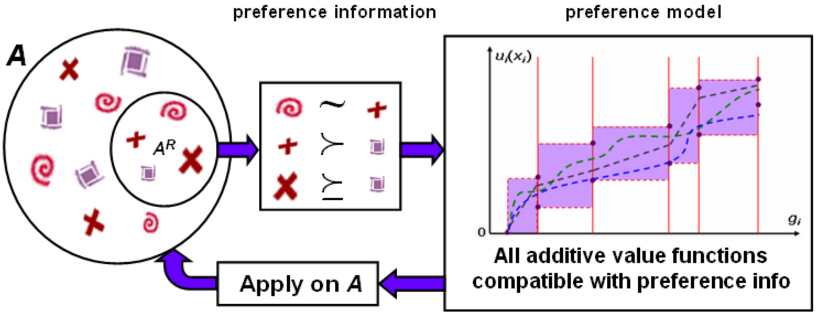
GPU computing



Questions before parallelizing

- Are you sure your code works?
- Implement speed-crucial parts of the code in lower level language (Java/C speedup over R/Matlab: 500x)?
- Does the implementation need to be used by others?
- Do the results need to be reproducible by reviewers?

Example 1: Parameter-set parallelization in LISA



- Computational experiments to assess the amount of preference inferences possible using the proven lemmas and theorems, and the amount of inferences still missing as we did not provide necessary conditions for the multiple statement case
- 600 test instances
- `https://github.com/tommite/pubs-code/tree/master/prefinf-ejor`

Tools of the trade: overview

- Generating the parameter set (cartesian product of sequences): **Python**
- Scheduling parallel execution of 7 processes in each node: **Python**
- Test script: **R** (some optimization through implementing certain methods in Java)
- Cluster scheduling: **Portable Batch System (PBS)**

PBS script

```
#PBS -l nodes=1:ppn=7 -l walltime=03:00:00 -t 1-86
python executeParallel.py $PBS_ARRAYID
```

- Schedule with:

```
qsub -d . effTests.pbs
```

Python script, page 1

```
import multiprocessing
from subprocess import call
import sys
from itertools import product

SCRIPT='test.R'
NRPROC=7
instId = sys.argv[1]
startIndex = (int(instId)-1) * NRPROC

## Script calling function
def callScript(parms):
        parStr = ' '.join(map(str, parms))
        print "Starting with parameters ", parStr
        call("R --vanilla --args " + parStr + " < " +
        SCRIPT, shell=True)
```

Python script, page 2

```
## Define the parameter set
alts = [10, 20, 50]
crit = [5]
nrpref = range(2, 41, 2)
instances = range(1, 11)

allTasks = list(product(alts, crit, nrpref, instances))
myTasks = allTasks[startIndex:(startIndex+NRPROC)]

## Start processing in parallel
pool = multiprocessing.Pool(processes=NRPROC)
print "Making computational tests for instance ID ", instId,
r = pool.map_async(callScript, myTasks)
r.wait() # Wait on the results
```

R script

```r
# ... Load libraries ...

### Read arguments
args <- commandArgs(trailingOnly = TRUE)
nalts <- as.integer(args[1])
ncrit <- as.integer(args[2])
npref <- as.integer(args[3])
instance <- as.integer(args[4])
stopifnot(length(args) == 4)
rm(args)
####
##

# ... Make actual tests ....
save('row', file=paste('fastror', nalts, \
 ncrit, npref, instance, sep='-'))
```

Output @ LISA

- Results for tests that completed within 3 hours
- For each job: effTests.pbs.o[ID]-[SUBID] (stdout) and effTests.pbs.e[ID]-[SUBID] (stderr)
- stdout file including SARA epilogue with information on job start and end times
- Copy results files to own computer and analyze results locally

Elementary tools @ LISA

- `qsub`: submit a PBS script
- `showq -u myuserid`: see your active, idle, and blocked jobs (limit of 4000h walltime)
- `qdel jobid`: remove job from scheduling (terminate with SIGTERM and SIGKILL if executing)
- `disparm`: another way to handle large parameter sets (not very handy IMO)
- `mpiexec`: mpi execution of jobs (more of this later)

Example 2: GPU parallelized importance sampling

The purpose is to simulate model parameters θ from the posterior density $f(\theta) \equiv f(\theta|\text{data})$:

1. $f(\theta)$ is not simple to simulate from $\Rightarrow$ Simulate M draws from $g(\theta)$ instead
2. Approximate function of interest $E(h(\theta))$:

$$E(h(\theta)) = \frac{\int h(\theta)\frac{f(\theta)}{g(\theta)}g(\theta)d\theta}{\int \frac{f(\theta)}{g(\theta)}g(\theta)d\theta} \approx \frac{\frac{1}{M}\sum_{i=1}^{M} h(\theta^{(i)})\omega(\theta^{(i)})}{\frac{1}{M}\sum_{i=1}^{M}\omega(\theta^{(i)})},$$

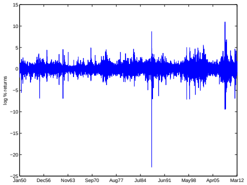
where $\theta^{(i)}$ is generated from a density with distribution $g(\theta)$.

- Evaluations $f(\theta)$ and $g(\theta)$ are independent $\Rightarrow$ direct parallelization is possible
- Computation of each $f(\theta)$ can be slow
- Might have to optimize the amount of data transfer

Importance Sampling applied to the GARCH(p,q) model

$$y_t = h_t^{1/2}\epsilon_t, \quad \epsilon_t \sim NID(0,1), \quad t = 1, \dots, T$$
$$h_t = \sigma^2 + \sum_{i=1}^{p} \alpha_i y_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}$$

S&P 500 data,
15657 observations.



Posterior density evaluation:

$$f(\theta|y) = \begin{cases} \prod_{t=\max(p,q)+1}^{T} \phi\left(y_t, 0, h_t\right), & \text{if } h_t > 0, \forall t \\ 0, & \text{otherwise} \end{cases}$$

$$h_t = \sigma^2 + \sum_{i=1}^{p} \alpha_i y_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}$$

where $\phi\left(y_t, 0, h_t\right)$ is the normal density pdf with mean 0, variance h_t.

Parallelizing GARCH(p,q) density evaluation

```
$\ln f(\theta) = -\infty$
if $\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1,\ \alpha_i > 0, \forall i,\ \beta_j > 0, \forall j$ then
    $\ln f \Leftarrow 0$
    $t^\star = \max(p, q)$.
    $h_1 = h_2 = \ldots = h_{t^\star} = \frac{1}{T} \sum_{t=1}^{T} y_t^2$
    for $t = (t^\star + 1) \rightarrow T$ do
        $h_t = \sigma^2 + \sum_{i=1}^{p} \alpha_i y_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}$
    end for
    for $t = (t^\star + 1) \rightarrow T$ do
        $\ln f = \ln f - \frac{1}{2} \ln(2\pi h_t) - \frac{1}{2} \frac{y_t^2}{h_t}$
    end for
end if
return $\ln f$
```

- Parallelization only possible on the MC iterations
- Speed gains possible through smart memory use

Speedups with different implementations

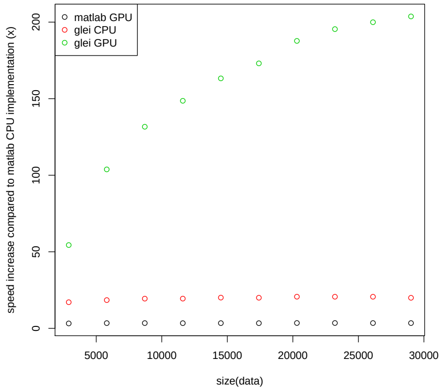


Figure : Instrumental variable model gleiCPU/GPU/MatlabGPU vs Matlab CPU

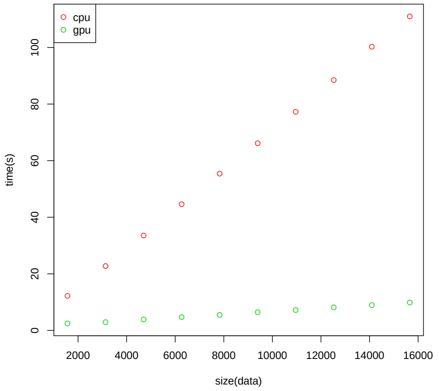


Figure : Garch(1,1) GPU vs CPU in glei

OpenMPI

- Message Passing Interface (MPI) library
- Processes have unique identifiers and can obtain the total amount of procs (compare with the earlier example)
- Processes belong to groups and can communicate with other processses in the same group
- Single node vs multiple nodes (multiple jobs with a limited amount of nodes get scheduled faster into execution)

Scientific communication relies on evidence that cannot be entirely included in publications, but the rise of computational science has added a new layer of inaccessibility.

[...]

We argue that, with some exceptions, anything less than the release of source programs is intolerable for results that depend on computation.

The vagaries of hardware, software and natural language will always ensure that exact reproducibility remains uncertain, but withholding code increases the chances that efforts to reproduce results will fail.

Ince, Hatton, and Graham-Cumming, Nature 482 (2012)

Q?

tervonen@ese.eur.nl