Multiprocessors
2014/2015

Parallel program design

Johan Lukkien
This is what we do....

Programming model: set of concepts, primitives and combinators to express a computation

machine model
(e.g. shared/distributed memory)
machine(s)

dots: alternatives
arrows: abstractions
(no arrow: abstraction not possible)

problem
designs [data, functional, result]
mapping
Overview

• Design of parallel programs
  – functional parallelism
  – data parallelism
  – result parallelism
Abstract problem statement

• Given:
  – initialized data structure \( D \)
  – function \( f \)

• Compute: \( f(D) \)

• **Note**: this represents an abstraction of just any algorithmic problem
  – cf. e.g. MapReduce
Functional parallelism

- Decompose $f$
  - ... and, hence, $D$
  - assign responsibilities for parts of the computation to processors
  - communication is induced

- Examples:
  - divide & conquer (e.g. merge sort, MapReduce)
  - pipelining (e.g. instruction parallelism, multimedia processing chain)

- Also called: “algorithmic decomposition”
- Fits a model of communicating processes
Example: MapReduce (Google, Apache)

- \( D \) is the Big Data
- Distributed according to an assignment by Master
- The ‘shuffle’ in the communication is induced by the partition function

Figure: data flow in MapReduce
Example: filters

- Block sorter
  - sort infinite input stream into blocks of size $N$

```
P_{Bubble}(\text{var } c?, d!: \text{port}) =
\langle \begin{array}{l}
\text{var } m, x: \text{int};
\text{while } true \text{ do }
\text{receive}(c, m);
\text{for } i := 2 \text{ to } N \text{ do }
\text{receive}(c, x);
\text{if } x > m \text{ then send}(c, m); \ m := x
\text{else send}(c, x)
\text{fi}
\text{od;}
\text{send}(c, m)
\end{array} \rangle
```

Idea: simple program: out of $N$ values hold up the highest value thus far encountered

Blocks of $N$ values

1st block

2nd block

$n$th block

$N$th block
Data parallelism

• Decompose \( D \)
  – and, hence, \( f \)
  – assign responsibilities for parts of \( D \) to processors
    • assignment pertaining to both results and input values
    • communication is induced, but usually regular

• Examples:
  – matrix multiplication, matrix solving
  – 2D FFT, multimedia computations
  – Molecular Dynamics simulation, finite elements computations

• Also called: “Domain decomposition”

• Typical issue:
  – distribution alternatives such as to minimize communication
  – synchronization and communication at domain boundaries
Example: matrix multiplication

```
for i := 0 to N-1 do
  for j := 0 to N-1 do
    C[i,j] := DotProduct (A[i,*], B[*,j])
```

- Each process(or) performs the computations required for the part of $C$ it is assigned to.
- To that end it may need to communicate locally stored parts of $A$ and $B$ to other processor, and receive parts of $A$ and $B$ it does not have.
Example distributions

- Striped
  - Row / column wise

- Blocked
  - more-dimensional stripes

- Cyclic

- .... + combinations
Striped

- \( P \) processors, \( NxN \) matrix
  - can also be just a *logical* arrangement of the processors

- Processor \( p \) is assigned rows \( p^* (N \text{ div } P) \) through \( (p+1)^* (N \text{ div } P) - 1 \)
Block

- \( P \) processors, \( N \times N \) matrix
- Striped in both rows and columns
Cyclic ("scattered")

- $P$ processors, $N \times N$ matrix

- Processor $p$ is assigned elements $(i,j)$ such that
  - $(i+j) \ mod \ P = p$

- Question:
  - which distribution is good (communication-wise) for
    - matrix multiplication
    - matrix transposition
  - can you use MapReduce as well?
    - what’s the difference?
Example: execution on NVIDIA GPUs

- GPU is a (collection of) SIMD
  - single instruction, multiple data
- Programmed using CUDA
  - Compute Unified Device Architecture
  - high/low level API: C-level API
    - control flow
    - specify concurrent operations on blocks

1. Copy data from main mem to GPU mem
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU mem to main mem
Result parallelism

• Apply \( f \) pointwise
  – \( f(D) = ( \cup d: d \in D: \{f(d)\} ) \)
  – extreme case of data parallelism

• Examples:
  – repeated executions of same tasks
    • getting good statistics in simulations
    • SETI
    • Google MapReduce for using Grep, Google File System
  – computing a picture on a per-scanline basis
    • e.g. fractals

• Also called: “processor farming”, “embarrassingly parallel”
Example organization

- Dashed lines: processor boundaries
  - Information stream split across two machines
  - Computation on two sub-streams concurrently
  - Results combined into one stream again.
Use a good idea twice...

Pipeline

Tree
Concluding remarks

• The three techniques may be applied together
  – e.g. computing the partial sums in a matrix multiplication in a pipelined or treelike fashion
  – and recursively...

• Different *granularity* levels should be balanced
  – e.g. if result parallelism is possible and single job latency is not the problem .... use it!
  – avoid communication ... it is overhead
    • hence, parallelism at highest possible level (largest grain size)
    • (extensive) distribution strategies should not take more time than the original computation...

• It may be possible that an inferior sequential algorithm is more amenable to parallel execution
  – e.g. Gauss-Jordan elimination