Talk at Royal Military Academy, Brussels, May 2004

Practical Parallel Processing

Jan Lemeire
Parallel Systems lab
Vrije Universiteit Brussel (VUB)
Example: Matrix Multiplication

\[ C = A \times B \]

\[ C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj} \quad (i, j : 1..n) \]

\[ T_{\text{computation}} = \delta_{mm} \cdot n^3 \]

Sequential algorithm
Parallel Matrix Multiplication

- **Parallel System**
- **Partitioning**

\[
p \text{ blocks of } \frac{n^2}{p} \text{ elements}
\]

- **Submatrix** \( C_{i,j} \):

\[
C_{i,j} = \sum_{k=1}^{\sqrt{p}} A_{i,\text{row }k} \cdot B_{\text{col }k,j}
\]

- **Communication**
Parallel Matrix Multiplication

Execution profile

Extra work = overhead

$n = 150$
Parallel Matrix Multiplication

Memory usage $\sim n^2$
Why Parallel Processing?

• Speedup \((time)\)
  – for long runs
  – realtime (eg. Simulations)
  – as much as possible (eg. weather forecasting)

• Memory Usage \((space)\)
Parallel Systems

Collection of
- Processors
- Memory
- Interconnection Network

1. Shared-Memory Architecture
   - fast communication
   - dedicated machines

2. Message-Passing Architecture
   - slower communication
   - simple, cheap
general-purpose PC’s
How? Communication Layer

- **Pvm** (Parallel Virtual Machine) or **MPI** (Message Passing Interface)
  - transparant
  - platform-independent

- **Functions**
  - create processes on other machines
  - send & receive messages
Aspects of Practical Parallelization

1. System-dependency
2. Inherent Parallelism
3. Software Engineering
4. Performance Analysis
1. System-dependency

- Network Topology

- Heterogeneous Systems
  - different processing powers
  - different communication speeds
  - combinations of shared memory & message passing architectures
2. Inherent Parallelism

- **Trivial parallelizable**
  - replicated trials (multiple experiments)
    => script
  - multiple jobs
    => job management
2. Inherent Parallelism II

- **Difficult to Parallelize**
  - Simulations
    - Synchronization protocol
    - Model dependent
  - Virtual 3D world
    - Tessellation, lighting calculations, rendering…

> Performance depends on various aspects, like data structures
> Optimizations are possible, but strongly depend on problem/algorithm
Example: Parallel Simulation

Detailed IP-switch

Execution profile

Parallel@RMA
3. Software Engineering

• Understandable, Maintainable
  – tangled code!

• Flexible
  – separate parallel code
  – Eg.: reuse sequential algorithm, so it can be adapted

• Reusable
  – trade-off generic program <> performance
4. Performance Analysis

- Detection of performance bottlenecks
  - For example
    - communication-computation ratio
    - load imbalances
- Scalability analysis
  - bigger problem => more computers
- Calculation of optimal number of processors
Performance Analysis Tools

- Automated analysis
  - Simple: XPvm
  - Complex
- However: Userfriendliness

=> EPPA
Our Performance Analysis Tool

1. Causal Models
to structure the relations between the variables
2. Refinement Strategy

Start: First-order approximation

\[ T_{\text{computation}} = \partial_{\text{comp}} \cdot \#\text{operations}. \]

Refine when necessary
Theoretical Conclusions

**Sequential world**
- Separation hardware – program (3GL)
  - With abstract model for architecture: Von Neuman
- Java: platform-independence
- .net: language-independence

**Parallel world**
Ultimate goal: match software - hardware

*No universal abstract model for parallel architectures!*

Conflict generality <> efficiency
- Performance is program- and hardware dependent
- Efficient programs should be developed specifically …
Practical Conclusions

- Successful parallel processing is a complex issue
  - But not ….
- Thus:
  - Is it worth it?
  - Is it possible?
  - Is it easy?
- Effort ~ Benefit