An Intelligent Information System for Parallelisation.

Jan Lemeire
August 2001

Thesis

An Intelligent Information System is necessary/usefull for parallelisation..

Def parallelisation
Communication, synchronisation, partitioning, blocking

Motivation

Parallelisation is a complex matter:

- It is problem-dependent, each problem requires a specific approach. There is no general solution that is performant for every problem.
- It is hardware-dependent, there is not something like a Von Neumann model for sequential processing, because a general hardware model looses optimality completely[cursus Erik].
- Parallelisation is about performance. Therefore generality and nice, structured solutions should be sacrified for optimisation.
- Parallel algorithms are complex. If something goes wrong for example, it is a hard job to find the error.

For succes however, parallelisation should be 'sold in a box'. A user should not worry about any aspect, parallelisation must work automatically for every hardware and every problem.
In my research I want to investigate how an intelligent information system (IIS) can tackle these problems. I will to try to proof that current techniques are extremely heavy, that it is not a matter of theory about parallelisation, but a matter of *how* to put this theory in a system.

Thus I should first define what a good parallel simulator is, what should be possible. Then analyse if present (programming) techniques meet these requirements and what intelligent techniques we need.instead. analyse hoe de technieken tegemoet komen aan de eisen (hoe noemt deze analyse??)
My research is thus mainly a thorough analysis and not a construction of another system, because I belief that criticical analysis is an important lack in informatics research.

Requirements

1. Automatic optimisation.	EXPR of rules
2. Automatic parameter detection.	RECOG
3. Performance prediction.	EXPR & RECOG
4. Automatic partitioning.	REAS
5. Algorithm reuse.	REUSE
6. Automatic algorithm selection.	
7. Model reuse.	
8. Debugging.	
9. Documentation.	LEVEL
10. Education.	

11. Simulation.

Current techniques

Current techniques for constructing information systems are mainly structured programming and OO. They aim flexibility by clarity and above all reuse, by generality. But then a first important question arises: don't we sacrifice performance for generality? Can we combine both or is it a tradeoff?
Secondly, do modern SE-techniques guarantee the necessary reuse and flexibility?
PROJECT: what is the cost of OO/C++ versus C?
Solution proposal: Optimal Code Generation!! (see LEVEL)

AI-techniques

When analysing the requirements of an ideal parallelisator, I will consider AI-techniques and evualuate them:

- expert system
- ontology
- NN
- GA

Intelligent Information System

My approach in defining an ideal, intelligent parallelisation machine is by applying the Turing Test: a human can write a perfect parallelised program, a computer should. With this question we derive the necessary properties.
(what humans can and computers should. For which jobs humans are still necessary)

To start my research I define following properties of information systems:

- **Expressiveness** (EXPR). An information system contains knowledge/information. Expressiveness is a measure for what kind of information can be put in the system. Especially the power of defining and applying general reasoning rules will be necessary for the parallelisator.
- **Recognition** (RECOG) of characteristics of the problem, like in pattern recognition. This relates to similarity detection (explained later).
- **Understanding**: for some requirements the information system should *understand what is meant*. However, this concept is not fully understood and still vague. A start of giving the concept a concrete form are the properies similarity detection (see recognition) and the notion of levels in a program (see later).
- **Minimal Information Principle**: hoe uitleggen? Uitwerken?

Position of Research

It is clear that it is a multidisciplinar problem. I belief that the fundamental problems can be detected I other areas as well and that the solution combines several research fields (this will be elaborated). Therefore a global approach is necessary and it can be considered as fundamental research.
Futhermore, there's good chance that this analysis will lead to a AI-hard problem, what justifies fundamental research.

Requirements Analysis

Performed as follow:

- Elaboration of examples.
- Evaluation of current techniques.
- Is an intelligent system necessary? Which (intelligent) properties are required.
- Evaluation of AI-techniques.
- Stipulate concrete reseach projects.

1. Automatic optimisation

Optimisations of an algorithm can go very far. Thus, the big question to investigate is to what degree optimisations can be performed automatic, like code automisation by the compiler.

Examples:

- Kumars optimisation of basic communication operations [Kumar, chapter 3] (PROJECT).
- In the parallel simulator: “Don’t sort an eventlist twice.” “Sort it only when it makes sense”. Optimisations that I performed on seqeager & misra.
- Eliminate an IF-test if you can predict the result.
- A calculated parameter that can be predicted. Eeg the windowsize, calculated by the null events, is same each cycle
- “Don’t calculate twice the same thing, remember the result using a variable.” Can such an optimisation rule be programmed?

Analysis:

- What kind of optimisations are there?
- What about model-dependent optimisations?
- What techniques are necessary?
  - For some optimisations *understanding* is necessary (Proof this!)

2. Automatic parameter detection

Parallelisation is problem-dependent. Therefore, problem characteristics should be known to optimise parallelisation. This should be automated.

Examples:

- Lookahead (PROJECT)
- Minimum, maximum delay of a process

Analysis:

1. some characteristics come immediately from model-specification (ok, automatic)
2. some can be measured, eg averages. (ok)
3. but for some, problem/code understanding is necessary (nok!)

For this last version, intelligence is necessary…

Idea:

Can we use the Compile-tree for code-understanding (PROJECT)??

Another thing is parameter *estimation*. This should ideally be done with a problem description, which will be informal and incomplete!

3. Performance prediction

I demonstrated in my paper [Lemeire 2000] that performance prediction is a matter of measuring or estimating certain problem characteristics. This problem can thus be reduced to the previous one (2. automatic parameter detection).
But how would a parallelisation expert estimate performance? He would use what is called “rules of thumb”, like estimating the communication/computation ratio. This resembles reasoning with fuzzy rules. So, is this possible for parallelisation.

4. Automatic partitioning

Partitioning should minimise parallel overhead (like communication and load imbalance). This task can be done in 2 ways. First, by exploring the space of all possible partitions (evt optimised search with genetic algorithms), this is a common search problem and is widely covered by current research. But this is not how humans do the partitioning. A search algorithm will not use symmetry properties of the problem, a human will. Is there a way of learning the parallelisator to use properties as symmetry?
Also in the problem of visualisation of the problem (giving processes a position)[many thesis’s deal with this], symmetry is a key issue, but automitisation of the concept fails

5. Algorithm reuse

Performance is problem-dependent. A good solution should therefore combine (all) research results. This is a nice example of a *reuse problem*. Can we use current reuse-techniques like OO? Remember that different labs use different simulators, different languages, different platforms, different approaches, … Can we overcome these difficulties? Can we make an interchange format for algorithm description (cf ontologies)?

6. Automatic algorithm selection

For optimal performance, the parallelisator should match problem and algorithm. Choose algorithm types and parameters in function of the problem characteristics (see 2). Can this flexibility be programmed with modern SE-techniques like OO and design patterns?

7. Model reuse

Models are written in different languages, for different simulators. How could we reuse them ? Build an interchange format for model description (see ontologies)?
Description of models (before formalisation) is in natural language, thus, ideally, models should be reused from natural language!
Furthermore, models are written in different levels of detail… (this will be considered in 11)

8. Debugging

er loopt iets fout… waar ligt de fout
je kan iets checken:

- niet dezelfde events seq & par
- er loopt iets fout met een event: van waar komt dat event, wat was de staat van het lp, etc (PROJECT: je kan vragen stellen)
- visueel voorstellen
- vragen kunnen stellen
- ??

9. Documentation

programs get complicated soon (algorithm is also complex), you forget why you program a scpecific code

ð explanation, links, levels (PROJECT)

cf SeqEager (kan je uit of aan zetten):

- TRACE
- DEBUG
- Optimisation(s)
- Uitleg => Nassi Schneider

OOK HIER: clear code!? Eg point 6, there will be optimisation all over the code!! How keep this overzichtelijk & duidelijk? Code generation!?

10. Education: LATER

Explain simulation, parallelisation, parallel algorithm…

11. Simulation: LATER

levels of detail