Evaluating the performance of skeleton-based high-level parallel programs

Enhance (funded by the EPSRC, grant number GR/S21717/01)

Enhancing the Performance Predictability of Grid Applications with Patterns and Process Algebras

A. Benoit, M. Cole, S. Gilmore, J. Hillston

School of informatics

http://groups.inf.ed.ac.uk/enhance/

Introduction - Context of the work

- Parallel programs in a heterogeneous context
  - run on a widely distributed collection of computers
  - resource availability and performance unpredictable
  - scheduling/rescheduling issues
- High-level parallel programming
  - library of skeletons (parallel schemes)
  - many real applications can use these skeletons
  - modularity, configurability $\rightarrow$ easier for the user
  - Edinburgh Skeleton Library eSkel (MPI) [Cole02]

Introduction - Performance evaluation

- Use of a particular skeleton:
  information about implied scheduling dependencies
- Model with stochastic process algebra
  - include aspects of uncertainty
  - automated modelling process
  - dynamic monitoring of resource performance

$\rightarrow$ allow better scheduling decisions and adaptive rescheduling of applications

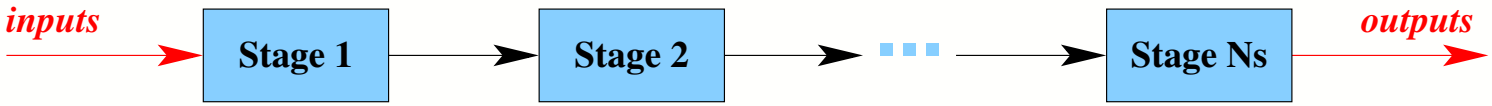
$\rightarrow$ *Enhance the performance of parallel programs*

Structure of the talk

- Introduction
- The Pipeline skeleton
  - Principle of the skeleton
  - Modelling the skeleton with PEPA [Hillston96]

    (Performance Evaluation Process Algebra)
- AMoGeT: The Automatic Model Generation Tool
  - Overview and input files
  - Numerical results
- Conclusions and Perspectives

Pipeline - Principle of the skeleton



- N_s *stages* process a sequence of *inputs* to produce a sequence of *outputs*
- All input passes through each stage in the same order
- The internal activity of a stage may be parallel, but this is transparent to our model
- Model: mapping of the application onto the computing resources: the network and the processors

Pipeline - Application model

- Application model: independent of the resources
- 1 PEPA component per stage of the pipeline $(i = 1..N_s)$

  $$Stage_i \stackrel{def}{=} (move_i, \top).(process_i, \top).(move_{i+1}, \top).Stage_i$$

- Sequential component: gets data ($move_i$), processes it ($process_i$), moves the data to the next stage ($move_{i+1}$)
- Unspecified rates ($\top$): determined by the resources
- Pipeline application = cooperation of the stages

  $$Pipeline \stackrel{def}{=} Stage_1 \underset{\{move_2\}}{\bowtie} Stage_2 \underset{\{move_3\}}{\bowtie} \ldots \underset{\{move_{N_s}\}}{\bowtie} Stage_{N_s}$$

- Boundary: $move_1$: arrival of an input in the application

  $move_{N_s+1}$: transfer of the final output out of the Pipeline

Pipeline - Network model

- Network model: information about the efficiency of the link connection between pairs of processors
- Assign rates λ_i to the $move_i$ activities ($i = 1..N_s + 1$)

$$Network \stackrel{def}{=} (move_1, \lambda_1).Network + \ldots$$
$$+(move_{N_s+1}, \lambda_{N_s+1}).Network$$

- λ_i represents the connection between the processor j_{i-1} hosting stage $i-1$ and the processor j_i hosting stage i
- Boundary cases:
  - j_0 is the processor providing inputs to the Pipeline
  - j_{N_s+1} is where we want the outputs to be delivered

Pipeline - Processors model

- Processors model: Application mapped on a set of N_p processors
- Rate μ_i of the $process_i$ activities ($i = 1..N_s$): load of the processor, and other performance information
- One stage per processor ($N_p = N_s \ ; \ i = 1..N_s$):

$$Proc_i \stackrel{def}{=} (process_i, \mu_i).Proc_i$$

- Several stages per processor:

$$Proc_1 \stackrel{def}{=} (process_1, \mu_1).Proc_1 + (process_2, \mu_2).Proc_1$$

- Set of processors: parallel composition

$$Processors \stackrel{def}{=} Proc_1 || Proc_2 || ... || Proc_{N_p}$$

Pipeline - Overall model

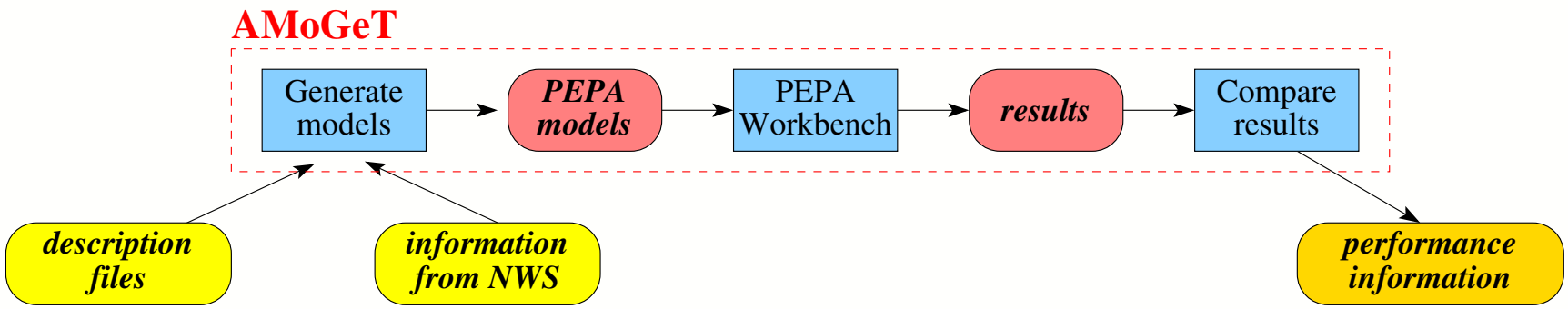
- The overall model is the mapping of the stages onto the processors and the network by using the cooperation combinator
- $L_p = \{process_1, ..., process_{N_s}\}$ synchronize $Pipeline$ and $Processors$
- $L_m = \{move_1, \ldots, move_{N_s+1}\}$ synchronize $Pipeline$ and $Network$

$$Mapping \stackrel{def}{=} Network \underset{L_m}{\bowtie} Pipeline \underset{L_p}{\bowtie} Processors$$

Structure of the talk

- Introduction
- The Pipeline skeleton
  - Principle of the skeleton
  - Process algebra model
- AMoGeT: The Automatic Model Generation Tool
  - Overview and input files
  - Numerical results
- Conclusions and Perspectives

AMoGeT - Overview



- AMoGeT: Automatic Model Generation Tool
- Generic analysis component
- Ultimate role: integrated component of a run-time scheduler and re-scheduler

AMoGeT

- Description file
  - timing information: time required to complete a stage on a processor, communication latencies, ... (some of these may be gathered from NWS)
  - mappings of stages to processors: location of the input data, the processor where each stage is processed, and where the output data must be left.

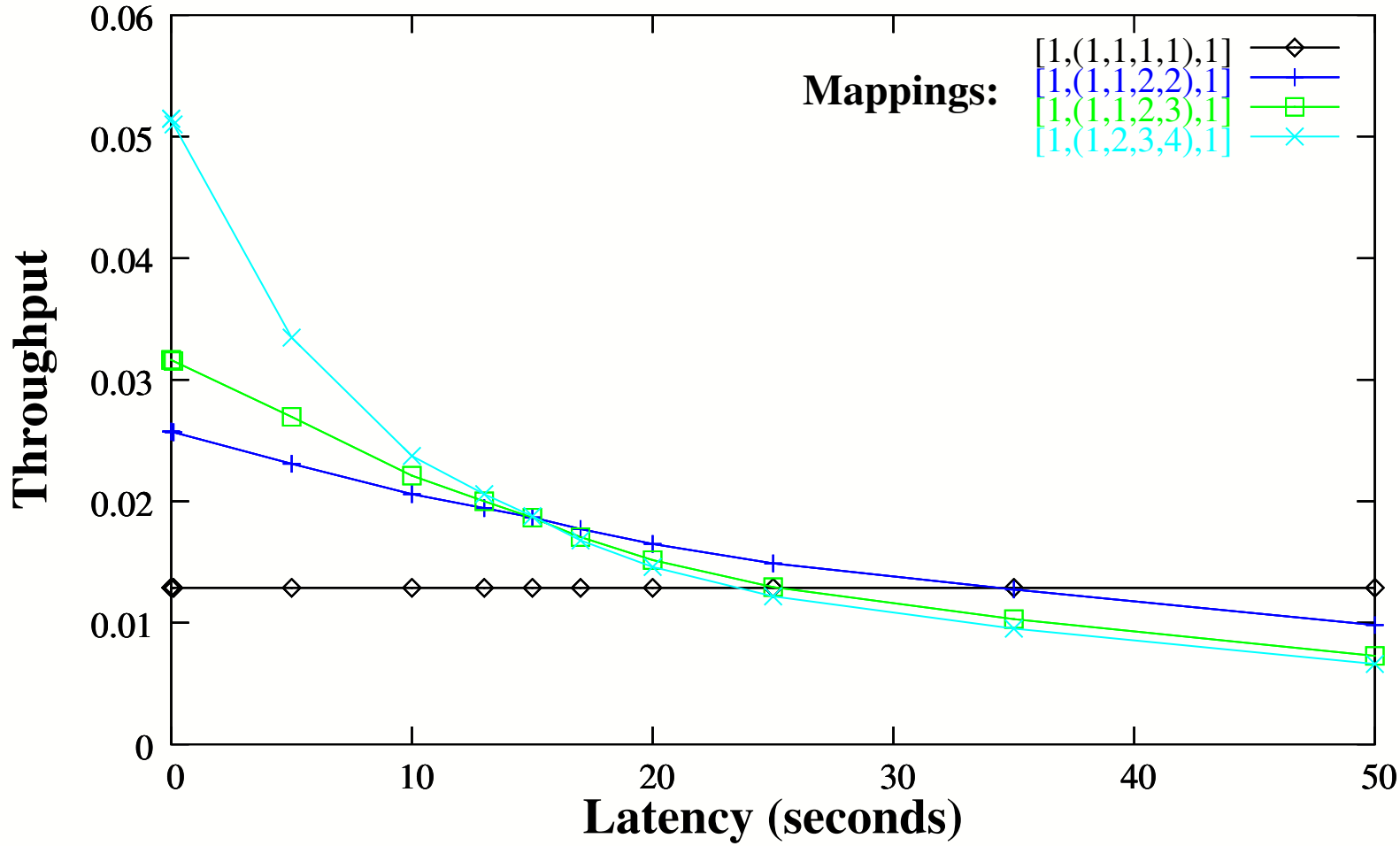
    ```
    mappings=[1,(1,2,3),1],[1,(1,1,1),1];
    ```
- Model generation straightforward (λ, μ)
- Models solved with the PEPA Workbench

Numerical Results

- Example 1: 3 Pipeline stages, up to 3 processors
- 27 states, 51 transitions $\rightarrow$ less than 1 second to solve
- latency of the com 0.001 sec; all stages/processors are identical; time required to complete a stage t
- $\mu_i = 1/(t \times nb_j)$ (nb_j: nb stages on processor j)
- Mappings compared: all the mappings with the first stage on the first processor (mappings $[1, (1, *, *), *]$)
  - $t = 0.1$: optimal mappings (1,2,3) and (1,3,2) with a throughput of 5.64
  - $t = 0.2$: same optimal mappings (one stage on each processor), but throughput divided by 2 (2.82)

Numerical Results

- Example 2: 4 Pipeline stages, up to 4 processors, $t = 10$



Conclusions

- Use of skeletons and performance models to improve the performance of high-level parallel programs
  - Pipeline skeleton
  - Tool AMoGeT which automates all the steps to obtain the result easily
  - Models: help us to choose the mapping to produce the best throughput of the application

Perspectives

- Provide more detailed timing information on the tool to prove its usefulness - *Recent work*
- Extension to other skeletons - *Deal*
- Experiments with a realistic application on an heterogeneous computational Grid

First case study → we have the potential to enhance the performance of high-level parallel programs with the use of skeletons and process algebras

Related projects

- The Network Weather Service – [Wolski99]
  - benchmarking and monitoring techniques for the Grid
  - no skeletons and no performance models
- ICENI project – [Furmento02]
  - performance models to improve the scheduling decisions
  - no skeletons, models = graphs which approximate data
- Use of skeleton programs within grid nodes – [Alt02]
  - each server provides a function capturing the cost of its implementation of each skeleton
  - each skeleton runs only on one server
  - scheduling = select the most appropriate servers

AMoGeT - Description files for NWS

- Specify the names of the processors
  - file `hosts.txt`: list of IP addresses
  - rank i in the list $\rightarrow$ processor i
  - processor 1 is the *reference processor*

```
wellogy.inf.ed.ac.uk
bw240n01.inf.ed.ac.uk
bw240n02.inf.ed.ac.uk
france.imag.fr
```

AMoGeT - Using the Network Weather Service

- The Network Weather Service (NWS) [Wolski99]
  - Dynamic forecast of the performance of network and computational resources
  - Just a few scripts to run on the monitored nodes
  - Information we use:
    - av_i fraction of CPU available to a newly-started process on the processor i
    - $la_{i,j}$ - latency (in ms) of a communication from processor i to processor j
- cpu_i - frequency of the processor i in MHz (`/proc/cpuinfo`)

AMoGeT - Generating the models

- One Pipeline model per mapping
- Problem: computing the rates
  - Stage s ($s = 1..N_s$) hosted on processor j (and a total of nb_j stages hosted on this processor):

$$\mu_s = \frac{av_j}{nb_j} \times \frac{cpu_j}{cpu_1} \times \frac{1}{tr_s}$$

  - Rate λ_s ($s = 1..N_s + 1$): connection link between the processor j_{s-1} hosting stage $s-1$ and the processor j_s hosting stage s: $\lambda_s = 10^3 / la_{j_{s-1},j_s}$

    (boundary cases: stage 0 = input and stage $N_s + 1$ = output)

AMoGeT - Solving the models and comparing the results

- Numerical results obtained with the PEPA Workbench [Gilmore94]
- Performance result: throughput of the $move_s$ activities = throughput of the application
- Result obtained via a simple command line, all the results saved in a single file
- Which mapping produces the best throughput?
- Use this mapping to run the application