Enhancing the performance of Grid Applications with Skeletons and Process Algebras

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

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

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 → 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

→ allow better scheduling decisions and adaptive rescheduling of applications
→ Enhance the performance of parallel programs
Structure of the talk

- The Edinburgh Skeleton Library eSkel
  and comparison with the P3L concepts
  - Motivation and general concepts
  - Skeletons in eSkel
  - Using eSkel
- Performance models of skeletons
  - Pipeline model
  - AMoGeT (Automatic Model Generation Tool)
  - Some results
- Conclusions and Perspectives
Concept of **skeletons** widely motivated

- **eSkel**
  - Murray Cole, 2002
  - Library of C functions, on top of MPI
  - Address issues raised by skeletal programming

- **eSkel-2**
  - Murray Cole and Anne Benoit, 2004
  - New interface and implementation
  - More concepts addressed for more flexibility
eSkel - Fundamental concepts

• Nesting Mode
  • define how we can nest several skeletons together

• Interaction Mode
  • define the interaction between different parts of skeletons, and between skeletons

• Data Mode
  • related to these other concepts, define how the data are handled

→ How do we address such issues in eSkel?
How are they addressed in P3L?
Can be either transient or persistent

- Transient nesting
  - an activity invokes another skeleton
  - the nested skeleton carries or creates its own data

- Persistent nesting
  - nested skeleton invoked once
  - gets the data from the outer level skeleton

- Linked to the data mode (detailed later)
- **Call tree built at the first interaction of each activity**
- **Structure of the persistently nested skeletons**
  - search in the tree to find interaction partners
- **Transiently nested skeletons**
  - not in the main tree
  - created dynamically, limited lifetime
  - subtree built dynamically when invoked
P3L (Anacleto, SkIE)
- all nesting of skeletons is persistent
- Defined within the P3L layer
- Clearly separated from the sequential code defining the activities

P3L-based libraries (Lithium, SKELib)
- Concept of transient nesting not explicitly addressed
- Not forbidden but not supported

ASSIST: not relevant
Can be either **implicit** or **explicit**

**Implicit**
- an activity has no control over its interactions
- function taking input data and returning output data

**Explicit**
- interactions triggered in the activity code
- direct calls to the generic functions *Take* and *Give*

**Additional devolved** mode for nested skeletons: the outer level skeleton may use the interaction mode of the inner skeleton
eSkel - Interaction Mode in P3L

- P3L and related libraries
  - Interaction via streams of data
  - Implicitly defined by the skeleton

- ASSIST
  - more flexibility
  - implicit or explicit interaction is possible
Related to the previous concepts

Buffer mode / Stream mode

- BUF: data in a buffer (*transient nesting*)
- STRM: the data flow into the skeleton from the activities of some *enclosing* skeleton call (*persistent nesting*)

**eSkel Data Model eDM**

- semantics of the interactions
- unit of transfer: *eDM molecule*
**eSkel - eSkel Data Model**

- **eDM molecule**: collection of **eDM atoms**
- **Type**: defined using standard MPI datatypes
- **eDM atom**: local versus global spread

![Diagram showing local versus global spread]

- Local Spread: 3 distinct items
- Global Spread: 1 item

P0

P1

P2

*abenoit1@inf.ed.ac.uk – Seminar in Pisa – 7th December 2004 – 13*
- **Pipeline & Farm**: classical skeletons, defined in a very generic way

- **Deal**: similar to farm, except that the tasks are distributed in a cyclic order

- **HaloSwap**: 1-D array of single process activities, repeatedly (1) exchanging data with immediate neighbours, (2) processing data locally, (3) deciding collectively whether to proceed with another iteration

- **Butterfly**: class of divide & conquer algorithms
Skeletons are commonly classified as

- **task parallel**: dynamic communication processes to distribute the work – *pipeline, farm*
- **data parallel**: works on a distributed data structure – *map, fold*
- **control skeletons**: sequential modules and iteration of skeletons – *seq, loop*

**eSkel**: only requires **task parallel** skeletons

- data parallel skeletons: use of the *eDM*
- control expressed directly through the C/MPI code
eSkel - Skeletons: interface

- eSkel:
  - not meant to be easy
  - based on MPI, the user must be familiar with it
  - structuring parallel MPI code

- P3L:
  - much easier to use, simple structure
  - less flexibility, structuring sequential code
  - data/task parallel and control skeletons
  - 3-stage pipeline: (create data, process, collect output)
void Pipeline (int ns, Amode_t amode[], eSkel_molecule_t * (*stages[])(eSkel_molecule_t *), int col, Dmode_t dmode, spread_t spr[], MPI_Datatype ty[], void *in, int inlen, int inmul, void *out, int outlen, int *outmul, int outbuffersz, MPI_Comm comm);

- general information about pipeline (ns, ...)
- specify the several modes: interaction mode (amode);
  data mode (dmode), spread (spr) and type (ty)
- information relative to the input buffer
- information relative to the output buffer
void Deal (int nw, Amode_t amode, eSkel_molecule_t *worker
(eSkel_molecule_t *), int col, Dmode_t dmode, void *in, int inlen,
int inmul, spread_t inspr, MPI_Datatype inty, void *out, int
outlen, int *outmul, spread_t outspr, MPI_Datatype outty, int
outbuffsz, MPI_Comm comm);

general information about deal (nw, ...)

specify the several modes: interaction mode (amode)
and data mode (dmode)

information relative to the input buffer

information relative to the output buffer
- Use of the library

- C/MPI program calling skeletons functions
- Great care should be taken for the parameters
- Definition of nested skeletons, workers, ... through standard C/MPI functions
- Only Pipeline and Deal implemented so far in eSkel version 2.0

→ Demonstration of the use of eSkel
Structure of the talk

The Edinburgh Skeleton Library eSkel
- Motivation and general concepts
- Skeletons in eSkel
- Using eSkel

Performance models of skeletons
- Pipeline model
- AMoGeT (Automatic Model Generation Tool)
- Some 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 \((i = 1..N_s)\)
  \[ Stage_i \overset{\text{def}}{=} (move_i, \top).(process_i, \top).(move_{i+1}, \top).Stage_i \]
- 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
  \[ Pipeline \overset{\text{def}}{=} Stage_1 \parallel Stage_2 \parallel \ldots \parallel Stage_{N_s} \]
- Boundary: \(move_1\): arrival of an input in the application
- \(move_{N_s+1}\): transfer of the final output out of the Pipeline
Network model: information about the efficiency of the link connection between pairs of processors

Assign rates $\lambda_i$ to the $\text{move}_i$ activities ($i = 1..N_s + 1$)

$$\text{Network} \overset{\text{def}}{=} (\text{move}_1, \lambda_1)\text{.Network} + \ldots$$

$$+ (\text{move}_{N_s+1}, \lambda_{N_s+1})\text{.Network}$$

$\lambda_i$ 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
Processors model: Application mapped on a set of $N_p$ processors.

Rate $\mu_i$ of the $\text{process}_i$ activities ($i = 1..N_s$): load of the processor, and other performance information.

One stage per processor ($N_p = N_s ; i = 1..N_s$):

$$\text{Proc}_i \overset{\text{def}}{=} (\text{process}_i, \mu_i) \cdot \text{Proc}_i$$

Several stages per processor:

$$\text{Proc}_1 \overset{\text{def}}{=} (\text{process}_1, \mu_1) \cdot \text{Proc}_1 + (\text{process}_2, \mu_2) \cdot \text{Proc}_1$$

Set of processors: parallel composition

$$\text{Processors} \overset{\text{def}}{=} \text{Proc}_1 \parallel \text{Proc}_2 \parallel \ldots \parallel \text{Proc}_{N_p}$$
The overall model is the mapping of the stages onto the processors and the network by using the cooperation combinator:

\[ L_p = \{ \text{process}_1, \ldots, \text{process}_{N_s} \} \text{ synchronize Pipeline and Processors} \]

\[ L_m = \{ \text{move}_1, \ldots, \text{move}_{N_s+1} \} \text{ synchronize Pipeline and Network} \]

\[
\text{Mapping} \overset{\text{def}}{=} \text{Network} \overset{L_m}{\boxtimes} \text{Pipeline} \overset{L_p}{\boxtimes} \text{Processors}
\]
AMoGeT: **Automatic Model Generation Tool**

- Generic analysis component
- Ultimate role: integrated component of a run-time scheduler and re-scheduler
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 - Description files (2)

- Describe the modelled application *mymodel*

  - file *mymodel.des*

  - stages of the Pipeline: number of stages $N_s$ and time $tr_s$ (sec) required to compute one output for each stage $s = 1..N_s$ on the reference processor

    $nbstage=N_s; \ tr1=10; \ tr2=2; \ ...$

  - 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.

    $mappings=[1,(1,2,3),1], [1,(1,1,1),1];$
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)
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 $\lambda_s$ ($s = 1..N_s + 1$): connection link between the processor $j_{s-1}$ hosting stage $s - 1$ and the processor $j_s$ hosting stage $s$: $\lambda_s = 10^3 / l_{a_{j_{s-1},j_s}}$

(boundary cases: stage 0 = input and stage $N_s + 1$ = output)
Numerical results obtained with the PEPA Workbench [Gilmore94]

- Performance result: throughput of the moves 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
Example 1: 3 Pipeline stages, up to 3 processors
27 states, 51 transitions → 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)
Example 2: 4 Pipeline stages, up to 4 processors, $t = 10$

Latency (seconds) vs. Throughput

Mappings:

- $[1,(1,1,1,1),1]$
- $[1,(1,1,2,2),1]$
- $[1,(1,1,2,3),1]$
- $[1,(1,2,3,4),1]$
Structure of the talk

- The Edinburgh Skeleton Library eSkel
  - Motivation and general concepts
  - Skeletons in eSkel
  - Using eSkel
- Performance models of skeletons
  - Pipeline model
  - AMoGeT (Automatic Model Generation Tool)
  - Some results
- Conclusions and Perspectives
Conclusions - Part 1

Why structured parallel programming matters? (Murray Cole’s invited talk to EuroPar 2004 in Pisa)

Presentation of the Edinburgh Skeleton Library eSkel

Concepts at the basis of the library

How do we address these concepts

Comparison with the P3L language and concepts, designed in Pisa.
**eSkel**: ongoing development and implementation phase

- Still several skeletons to implement
- Interface could be made a bit easier and user-friendly
- Necessity of a debugging mode to help the writing of application (checking the correctness of the definitions in the application code, the coherence between modes, ...)

→ **Demo for interested people**
Validation of these concepts

- Develop a real application with eSkel
- Promote the idea of skeletons

Comparison with other approaches

- P3L, ASSIST, ...
- Kuchen’s skeleton library
- Parallel functional language Eden

→ Motivation for my visit in Pisa
Use of skeletons and performance models to improve the performance of high-level parallel programs

- **Pipeline** and **Deal** 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
- Use of the **Network Weather Service** to obtain realistic models
Provide more detailed **timing information** on the tool to prove its usefulness - *Recent work*

Extension to **other skeletons**

Experiments with a **realistic application** on an heterogeneous computational **Grid**

Integrate in a **graphical tool** to help the design of applications with **eSkel**

*First case study* → we have the potential to enhance the performance of high-level parallel programs with the use of skeletons and process algebras
Thank you for you attention!

Grazie per la vostra attenzione!

Any questions?
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