

IST ACOTES Project
Deliverable D3.3

Cost Model Implementation

IST-034869

Public

Project Number	:	IST-034869
Project Title	:	Advanced Compiler Technologies for Embedded Streaming
Deliverable Type	:	Demonstrator

Deliverable Number	:	D3.3
Title of Deliverable	:	Cost Model Implementation
Nature of Deliverable	:	Demonstrator
Internal Document Number/version	:	acotes-d3.3-final
Contractual Delivery Date	:	30 August 2008
Actual Delivery Date	:	8 September 2008
WP(s)	:	WP 3 High level transformation parameterised by ASM
Author(s)/Affiliation	:	Paul Carpenter/UPC

Abstract

This document describes the usage of the cost model used in the ACOTES project to steer the compilation process

Keyword list

ASM Abstract Streaming Machine
Cell Cell Broadband Engine

Table of Contents

Table of Contents.....	3
1 Introduction.....	4
2 Cost model simulator.....	5
3 Getting started with the simulator.....	6
4 Regression test.....	7
5 Using the interactive viewer.....	8
6 Contents of the system definition file.....	9
7 Basic implementation structure of the simulator.....	10
8 Examples.....	11
8.1 System Definition Files.....	11
8.2 Streaming programs.....	11
8.3 Example hardware platforms.....	13
9 Generating graphical output.....	15

1 Introduction

This report describes the use of the cost model that is used to steer the compilation process in the ACOTES project.

The cost model is implemented as a simulator.

2 Cost model simulator

The directory of the cost model simulator contains the following files:

README	This file
sim.py	Simulator
viewer.py	Interactive viewer
test.sh	Regression test
examples/	Example system definitions
platforms/	Example hardware platforms
programs/	Example programs
ref/	Reference output

3 Getting started with the simulator

As the simulator is written in Python, you must have Python installed on your system. You will need at least Python 2.4, which has support for set types. I am using Python 2.5.

From the project root directory, you may run the simulator using a command such as:

```
./sim.py -n 30 examples/square_on_mesh2x2.py
```

This should run the simulation for 30 cycles and generate a simple trace of the execution on standard output.

The 'irregular' example is of a system that reaches a steady state of period six iterations:

```
./sim.py --period -n 2000 examples/irregular.py | tail -30
```

The usage information is as follows:

```
Usage: sim.py [options] system-defn-file
Options:
  --help                Display this information
  --internal            Display internal information
  --num <arg>          Number of timesteps to simulate
  --numiters <arg>     Number of iterations (of last task) to simulate
  --paraver <filename> Generate output for Paraver
  --local              Show local communications in Paraver trace (default off)
  --period             Generate information about length of periods
  --periodtotals       As above, but show totals rather than increments
  --statistics         Display statistics about steady state; etc. when complete
  --override <statement> Perform statement to override definition of system
  --first <arg>       Suppress output before given timestep (SimpleTrace)
  --tracein <filename> Process an input Paraver trace
  --dot               Generate .dot format output files only
  --seed <value>     Set seed for random number generator
```

4 Regression test

Run the regression test using the command:

```
sh test.sh
```

You should see the output:

```
testing square_on_mesh2x2...
testing square_on_smp2...
testing fir_on_line...
testing fibonacci...
testing irregular...
testing multihop...
testing rateconv...
testing str2hash_on_cell...
testing tolower_on_acc...
testing gnuradio_on_cell...
Running unit tests using doctest...
Test passed
```

The output files will be placed in the out/ directory.

5 Using the interactive viewer

The interactive viewer, `viewer.py`, shows the steady-state iteration, as generated by

```
sim.py -statistics
```

and allows various parameters to be updated on-the-fly.

For example:

```
./viewer examples/irregular2.py
```

Shows the steady state for the `irregular2` example, which begins:

```
Time      p (A)      pq(X)      q (B)      qr(X)      r (C)
-----
t= 1270  Processing 46  -          (PopWait 43)  Gap 39  ->PopWork 39
t= 1271  Processing 46  -          (PopWait 43)  Gap 39  PopWork 39
t= 1272  Processing 46  ->Start 43  (PopWait 43)  -       Processing 39
t= 1273  Processing 46  Start 43   (PopWait 43)  -       Processing 39
t= 1274  Processing 46  Transfer 43 (PopWait 43)  -       Processing 39
t= 1275  Processing 46  Transfer 43 (PopWait 43)  -       Processing 39
t= 1276  Processing 46  Transfer 43 (PopWait 43)  -       Processing 39
t= 1277  Processing 46  Transfer 43 (PopWait 43)  -       (PopWait 40)
t= 1278  Processing 46  Transfer 43 (PopWait 43)  -       (PopWait 40)
t= 1279  Processing 46  Transfer 43 (PopWait 43)  -       (PopWait 40)
...
Increase  q          w:e        r          t:y        u
Decrease  a          s:d        f          g:h        j
```

You may scroll the screen using the vi-like commands `Ctrl-F`, `Ctrl-B` and the up and down arrow keys. You may precede a key by a single digit in the range 2 through 9, to repeat the effect of the key that many times.

The bottom of the screen shows how to modify various parameters of the system, for example:

- press the 'a' key twice to reduce the processing time for task pq from 25 to 23 cycles.
- press the 'q' key to increase the message size between p and q by one element. Depending on the size of each element and the bandwidth of the link, doing so once may not make a difference.
- type the sequence "9d9d" to reduce the queue length between p and q by 18 elements, from 24 to 6, thus causing a deadlock (as a single message contains 7 elements).

6 Contents of the system definition file

The system definition file should provide the following functions:

<code>setup_program</code>	Returns a Program, which defines the set of Tasks and Streams in the streaming program.
<code>setup_platform</code>	Returns a Platform, which defines the set of Processors and Links in the hardware.
<code>setup_system</code>	Returns a System, which maps Tasks to Processors, provides explicit static routing information if required, and defines the length of the queue for each Stream.

See the examples for more information.

7 Basic implementation structure of the simulator

The source files are as follows:

<code>sim.py</code>	Parses command line options and runs the outer loop of the simulator
<code>Engine.py</code>	Contains the machinery to run the event-driven simulation
<code>Program.py</code>	Defines the behaviour of a streaming program, comprised of Tasks and Streams
<code>Platform.py</code>	Defines the behaviour of the hardware platform, comprised of Processors and Links
<code>System.py</code>	Defines the behaviour of the system, comprised of Threads (one per processor) and Edges (zero or more hops per stream).
<code>GraphOutput.py</code>	Generate .dot files for the program and platform
<code>Trace.py</code> <code>SimpleTrace.py</code> <code>StatisticsTrace.py</code> <code>ParaverTrace.py</code> <code>PeriodTrace.py</code>	Generate output in the appropriate format
<code>Util.py</code>	Utility functions
<code>Enum.py</code>	C-style enumeration class

8 Examples

8.1 System Definition Files

These files can be used directly by the simulator.

```
fir_on_line
  FIR program on a line of three processors

tolower_on_acc
  tolower program on a CPU plus accelerator

multihop
  Example system with non-local communication

rateconv
  Example system with rate conversion

square_on_mesh2x2.py
  Square program on a 2D mesh

square_on_smp2
  Square program on a 2-way SMP

tolower_on_acc
  tolower program on a CPU plus accelerator

gnuradio_on_4cpu
  GNU radio program on 4 CPUs

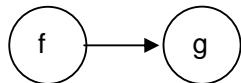
irregular
irregular2
  Simple systems demonstrating a period greater than one iteration, due to contention
  on a shared bus.

fibonacci
  Simple example of self-feedback
```

8.2 Streaming programs

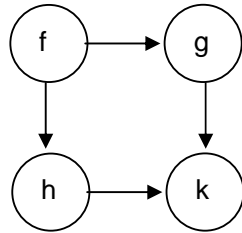
These programs, in directory examples/programs, may be used by the system definition files.

```
producer_consumer
  Two tasks communicating in the following pattern.
```



Square

Four tasks, communicating in a square pattern.



tolower

Three tasks communicating in the following pattern.



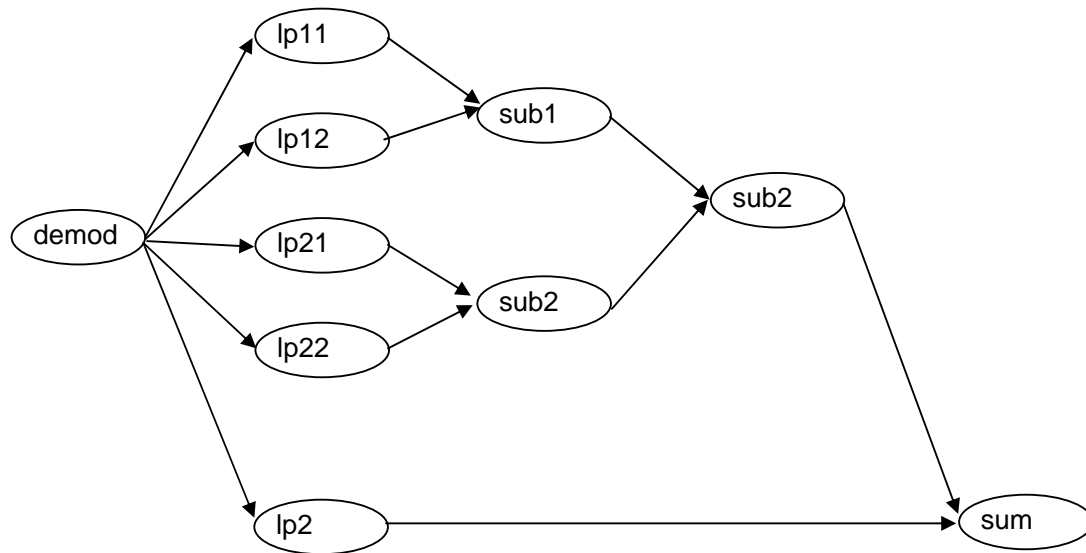
Fir

Finite Impulse Response filter, showing how to have different init and work functions.



gnuradio

GNU radio example

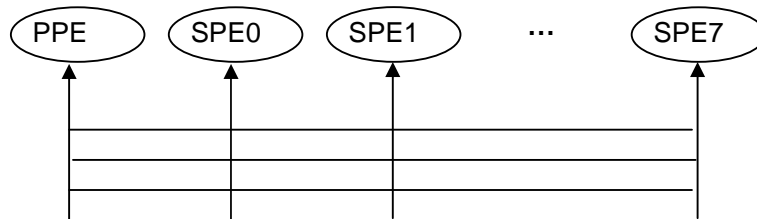


8.3 Example hardware platforms

These platforms, in directory examples/platforms, may be used by the system definition files.

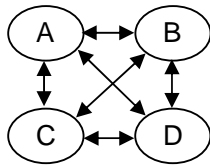
`cell`

Simple model of the Cell processor, using n (default four) ordinary buses to model the EIB (Element Interconnect Bus), which is actually a ring.



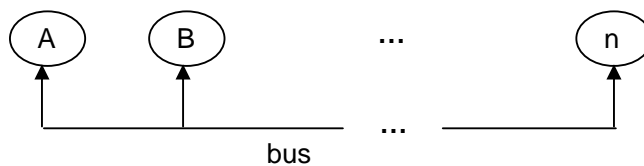
`complete`

n processors, with a dedicated link between each pair of processors. By default $n=4$, but this can be overridden.



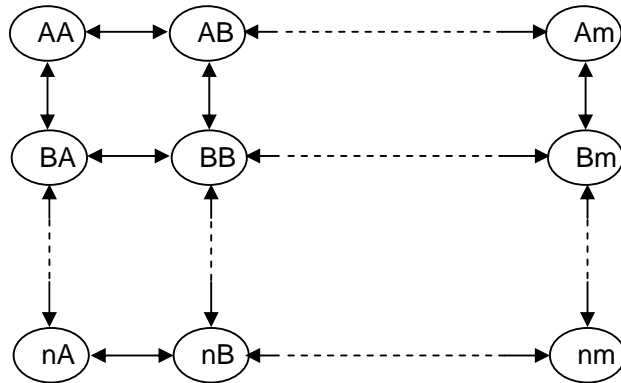
`smp`

Symmetric Multiprocessor with a single shared bus and n processors. By default $n=16$, but this can be overridden.



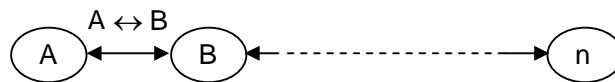
mesh2d

Two-dimensional mesh with $n * m$ processors. By default $n=m=2$, but this can be overridden.



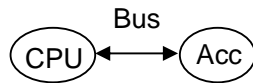
Line

Line topology with point-to-point links between adjacent processors. The default number of processors is $n=16$, but this can be overridden



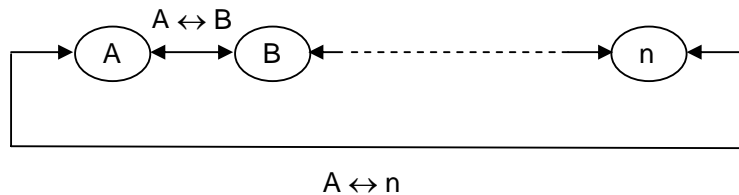
acc

Single CPU plus accelerator - same as smp2, except for the names given



ring

Ring topology with point-to-point links between adjacent processors. The default number of processors is $n=16$, but this can be overridden



9 Generating graphical output

Using the `--dot` option, it is possible to generate simple graphs representing the program, platform and system.

The command line:

```
./sim.py --dot examples/square_on_mesh2x2.py
```

should generate three new files in the current directory (and not run a simulation):

```
square_on_mesh2x2.platform.dot  
square_on_mesh2x2.program.dot  
square_on_mesh2x2.system.dot
```

These files are in the DOT language, and may be viewed, for example, using GraphViz, available from www.graphviz.org. As they just define the topology, you may need to tweak them to make them look good.