Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-6-2007

Synthetic Coupled Workload Models for the Bespoke Framework Generator and MPI

Egan,G.K., Riley, G.D., Ford, R.W., and Armstrong, C.

Synthetic Coupled Workload Models for the Bespoke Framework Generator and MPI

G.K. Egan[1], G.D. Riley[2], R.W. Ford[2], and C. Armstrong[2]

Department of Electrical & Computer Systems Engineering[1]
Monash University
Melbourne, Australia.

Centre for Novel Computing[2]
School of Computer Science
University of Manchester, England
Manchester, England

Abstract— **This paper describes a synthesiser for synthetic computational coupled-models. The Synthesiser produces input metafiles for variants of the Bespoke Framework Generator as well as a freestanding MPI implementation.**

***Index Terms*—coupled-models, synthetic, workloads, MPI, BFG.**

I. INTRODUCTION

COUPLED-MODELS are comprised of two or more entities that exchange messages. The message exchanges need not be periodic. In many cases the communications may be asynchronous, that is, not locked to a particular time, and usually result in a change in the internal state of the communicating entities.

It may be argued that all natural systems act like coupled-models. Communication between entities is certainly asynchronous with communication rates depending upon the rate of change of local state. Communication is usually short-range with the effects of local state change propagating with some latency across the large collection of entities. The coupled-models may often be hierarchical or at least viewed as such.

More recently there has been an interest in interconnecting existing legacy codes, each modeling some facet of a physical system, to obtain a better understanding of the overall system's behaviour. Global weather modeling, bringing together atmospheric, ocean and surface codes amongst others, is but one example.

There is usually a profound reluctance to modify the constituent legacy codes some elements of which were developed decades ago and may often exhibit some fragility and occasionally incompatible science. The fragility is arguably due to later authors.

The Centre for Novel Computing at Manchester has been involved for some time in the development of frameworks to support coupled-models and their deployment. The Bespoke Framework Generator (BFG) was developed to support these studies.

Unfortunately there are relatively few significant coupled-models available to researchers. In many cases those that exist are not generally accessible for extended studies on scheduling (static and dynamic) and the use of heterogeneous distributed computing platforms. This is usually because of IP issues associated with particular codes. BFG recognizes this be requiring only the interface specifications to build the harness code for a coupled_model.

This paper describes a synthesiser for workloads intended to support extended studies in coupled-models including dynamic workload scheduling and the scalability of coupled-models generally.

II. THE BESPOKE FRAMEWORK GENERATOR

BFG, using a set of metafiles, produces harness code linking several legacy codes allowing them to exchange information periodically. The harness conditionally calls subroutines within the legacy codes directly. The metafiles specify which codes are to be used, the subroutine entry points, how the codes are connected and the types of the data to be exchanged.

It is not our intention here to detail BFG as it is covered adequately elsewhere [1-7].

The clear intention is not to alter the original legacy codes in any way. However coupling these codes often discloses unstructured initialization and/or reliance on side effects in the codes as presented.

III. TERMS

The terms outlined here attempt to cover those most likely in coupled-models with synchronous periodic communications and are used in various BFG variants.

A <coupled_model> is comprised of two or more <model>s.

- Each model may communicate periodically with one or more other <successor> models but not with itself directly, this in any case being unnecessary as each model preserves its internal state. For any given successor this period is called the <coupling_period>.

- Models commence executing after some initial number of cycles or <offset>.
- Models are enclosed within an overall loop or <iteration>; completion of one iteration constitutes a <cycle>. Some models may not execute every iteration but every fixed number of cycles or <model_interval>.
- A communication may be specified to arrive at its successor(s) after some <lag> from the current iteration.
- The graph will repeat its pattern of communications after some number of iterations. This is termed a <major_cycle>.
- The communication links between models may, or may not, be <primed>. Priming and the priming sequence must consider the possibility of deadlock. Priming occurs in the <initialization> phase outside the main iteration.
- Finally the coupled-model after completing the requisite number of iterations performs <termination>. Some coupled-models may be continuous and so there is no <termination> as such but there may be break-pointing which could be in the form of a priming-set of messages and captured model-state for a restart.

In what follows the offset is set to zero.

IV. BFG Variants

It may be seen that if the model_intervals of communicating pairs of models and the associated coupling_periods and lags of the connecting links expressed in iterations do not follow a ‘convenient’ modulo inter-relationship then models may communicate infrequently or possibly never!

To make the Synthesiser a little more rational we restrict the models generated as follows.

There are a number of ways that the main iteration may be generated under BFG. One form has an iteration corresponding to the least frequently executing model and plants loops within the main iteration to iterate some models at higher rates corresponding to their model_intervals. Alternatively the iteration is determined by the most frequently executing model with others being executed modulo their model_interval. We chose the latter scheme for the Synthesiser.

A. BFG2

BFG2 currently uses only model_interval. Models execute if:

$$(\text{iteration mod model\_interval})=0 \qquad (1)$$

Models execute every model_interval regardless of whether they receive new messages from other models.

They receive messages from predecessor models on any given input if:

$$((\text{iteration}-\text{successor\_model\_interval}) \text{ mod } \text{predecessor\_model\_interval})=0 \qquad (2)$$

Models send data to successor models if:

$$(\text{iteration mod successor\_model\_interval})=0 \qquad (3)$$

To do this their firing condition (1) must also be satisfied.

B. BFGx

For BFGx we assume models execute when messages arrive at one or more of their inputs. At least one model is assumed to execute on the first iteration.

Messages are sent or received if:

$$(\text{iteration mod coupling\_period})=0 \qquad (4)$$

We have chosen here not to use model_interval to further condition which messages are received or transmitted. The more general consequence of this is that if a model receives one or more communications in a particular iteration it becomes active in turn sending messages to successor models. This may be viewed as a more data-driven coupled-model with explicit lag in each communication path. The model_interval for any given model varies depending upon arriving messages.

V. The Workload Synthesiser

The Synthesiser generates BFG2, BFGx coupled-models and simple flat models. Flat models are directed graphs with no iterations.

The directory tree of files produced contains:

1. f77 Model sources and associated Makefiles.
2. BFG metafiles for:
   - models,
   - composition,
   - coupling, and
   - default deployment
3. An unrolled directed acyclic graph, of nominally one major cycle in the case of BFG2. For BFGx the length of a major_cycle is not immediately obvious. For flat graphs it is the entire graph.
4. An MPI f77 compilable source.

Examples of the files generated are given in the Appendix. The default deployment is MPI with all models executing conditionally on every iteration. There are no ordering dependencies between the models for any given iteration.

A. Synthesis process

The overall process is relatively straightforward consisting of producing models with random memory-footprints and execution times interconnected randomly to other models. The interconnections must observe the constraints of the BFG specification [2].

a) Models

Each model has three subroutines corresponding to the initialization, iteration and termination phase. The subroutines are generated from a simple f77 template.

The memory-footprint and run-time, chosen randomly, are assumed (perhaps unrealistically) to be uncorrelated. The memory-footprint is exercised by indexing a vector of size memory_footprint cyclically modulo the loop variables controlling the run-time (See Appendix). Timers are not used as target processors may have varying performance and for scheduling it is necessary that models do see a reduced execution time on faster processors. Strictly run-time should be seen as a normalized workload.

b) Coupling

The number of successor models for each model and the coupling_periods associated with each connection are chosen randomly. Models are not connected directly to themselves.

Message sizes are chosen at random and again are assumed to be uncorrelated to memory-footprints or run-times. Data in the messages is not used by the synthesised models other than to propagate a distributed checksum.

B. DAG

The coupled-model is unrolled as a directed acyclic graph (DAG) for a major-cycle with an initial dummy source model and final dummy sink model inserted to form a complete DAG as required by some scheduling techniques. These have no associated computation cost. The numbering of the unrolled models is systematic permitting the original model's identity to be determined.

The DAG is intended to be used to produce a static or at least initial schedule to guide BFG's deployment generation or other scheduling studies.

A constant communication rate between models is assumed, which is reasonable given that the schedule is not known to the Synthesiser. As the Synthesizer also does not have information on relative processor performance it produces a cost file that assumes equal performance.

The communication and processor cost files generated can be rescaled to reflect the actual relative processor performance and inter-processor communication cost. It is anticipated that this would be drawn from a database of previously measured performance. The Synthesiser may be modified easily to incorporate database information but for now normalized performance is deemed acceptable.

Files compatible with the "dot" graphical display program from the Graphviz suite [8] are produced to allow visualisation of the DAGs. One form shows models as simple bubbles while the other shows the models as rectangles with their height scaled to run-time.

Artificial dependencies, between invocations of each particular model, have been introduced to reflect the order imposed by the model_intervals for BFG2; they have no communication cost.

C. MPI

The MPI versions of the synthesized coupled-models are intended to be independent of BFG and act as a reference for automatically generated control and communications code produced by BFG Synthesisers.

MPI wrappers are generated that receive messages, call the associated model subroutine, and conditionally send messages. A root MPI program invokes the models in a containing iteration for BFG or directly in the case of a flat execution model.

D. Synthesiser parameters

The Synthesiser takes a single line of input data from standard input. For most parameters the numeric value is the upper bound of an interval from which the actual parameter is chosen at random. The fields are as follows:

- <model type> The type of model to be synthesized;{0=bfg2, 1=bfgx, 2=flat}.
- <fully connected> If set then all models will have at least one input or output connection;{0=may not be fully connected, 1=fully connected}
- <number of processors> This is the number of processors on which the final model will be expected to run.
- <scale> The Synthesiser builds models of a particular run-time based on and assumed base processor performance corresponding to a Dell desktop dual core Pentium circa 2007. If the actual processor is twice as fast then the scale should be set to 2.0
- <models> The number of models comprising the coupled model excluding and dummy source or sink models. Models are named from m1 upwards.
- <model interval> This is the maximum model interval.
- <fanout> The maximum number of successor models for any given model.
- <coupling period> The maximum coupling period.
- <lag> The maximum lag included for completeness as the Synthesiser sets lag to zero.
- <run-time> The maximum run-time in seconds for any model assuming the Pentium base processor.
- <memory-footprint> The size of the memory-footprint in 32bit words that synthesized models will index cyclically.
- <message size> The maximum message size in 32bit words that will be sent between models.
- <plot rectangles> DAG representations in "dot" format; {0=circles, 1=rectangles scaled to run-time}.

There is some limited checking of parameter validity. Because of the random generation process some graphs may be sparsely connected.

Setting the fully connected parameter results in the Synthesiser attempting to generate models where there is at least one connection to a model when it is executed. If it is not set then the Synthesiser may still produce a connected model; it simply does not check.

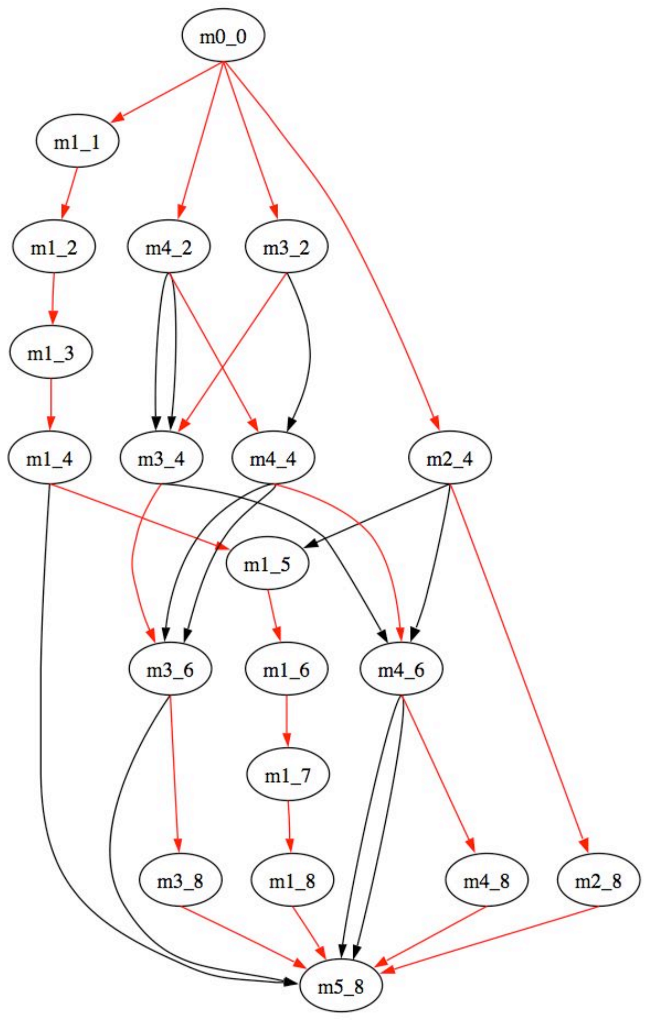
Re-running the Synthesiser will produce another (different) coupled-model for the same input specification. The maximum number of models comprising the coupled_model is 200.

VI. Examples

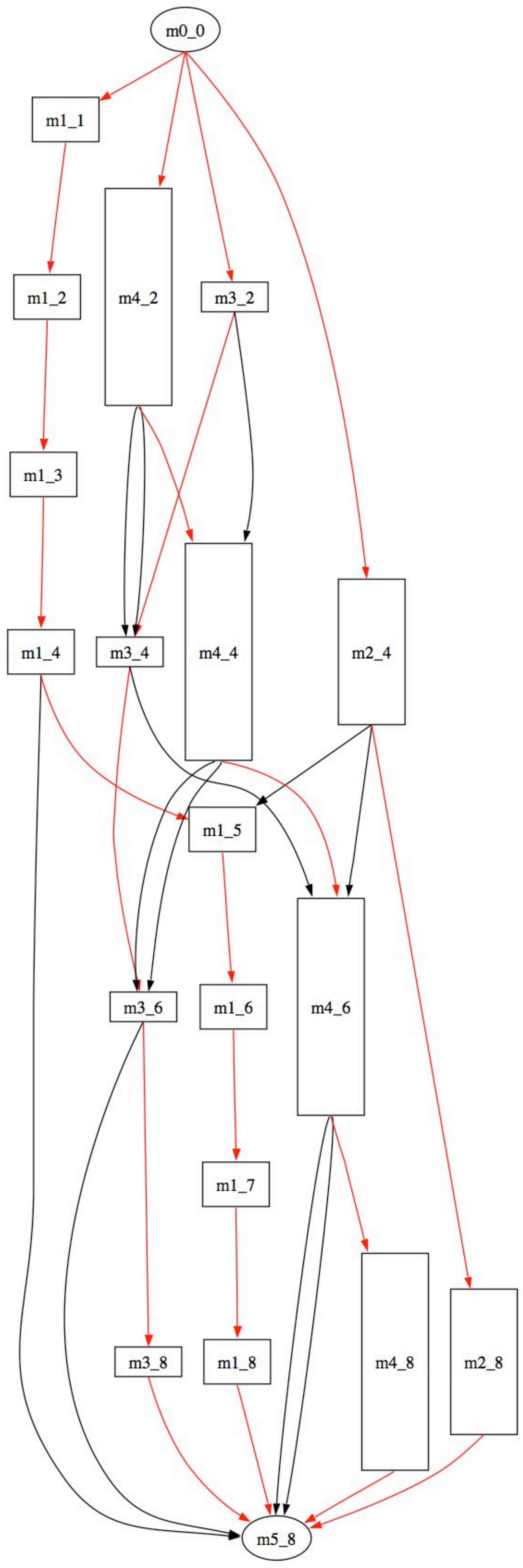
A number of examples follow. Each is annotated with its associated specification file. The communication weightings on arcs in these graphs as they only serve to clutter.

A. BFG2 Coupled-model with Four Models

In this case both time-scaled and unscaled versions are shown. Note that although a connected model has been selected it does not as one may expect guarantee that the models are fully connected, just that there is at least one connection.



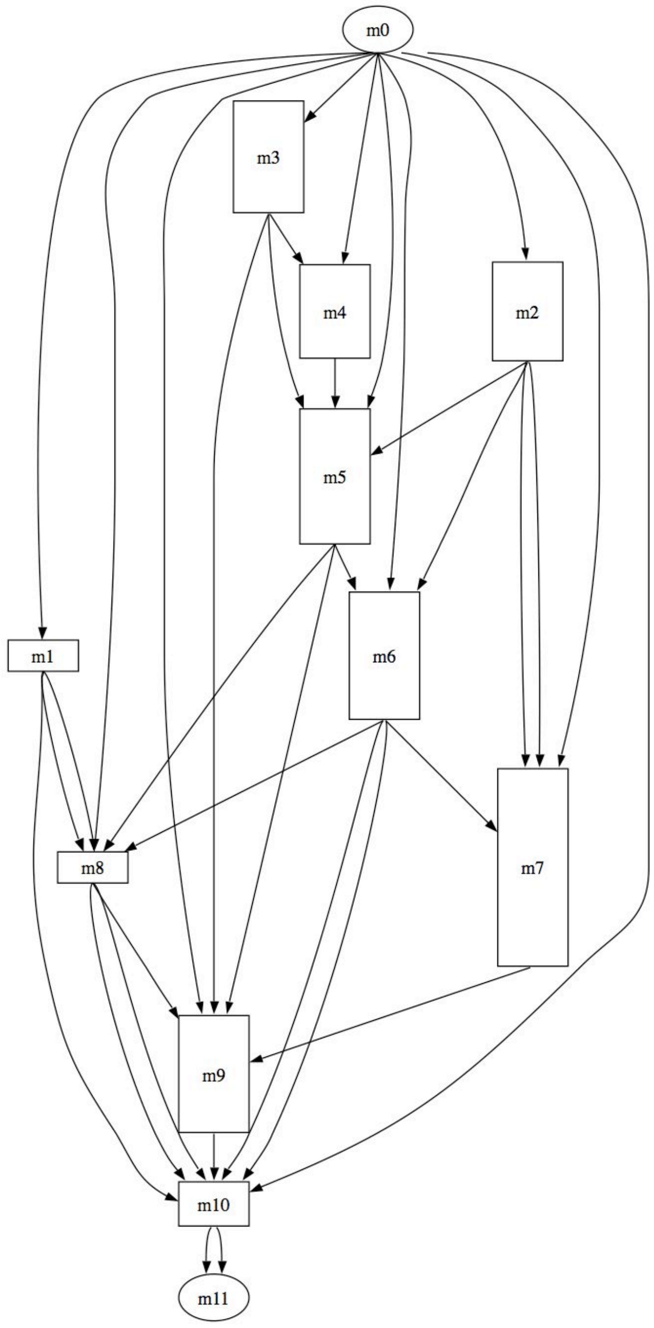
0 1 4 1.0 4 5 2 5 1 5.0 100000 10000 0



0 1 4 1.0 4 5 2 5 1 5.0 100000 10000 1

B. *Short Ten Model Graph*

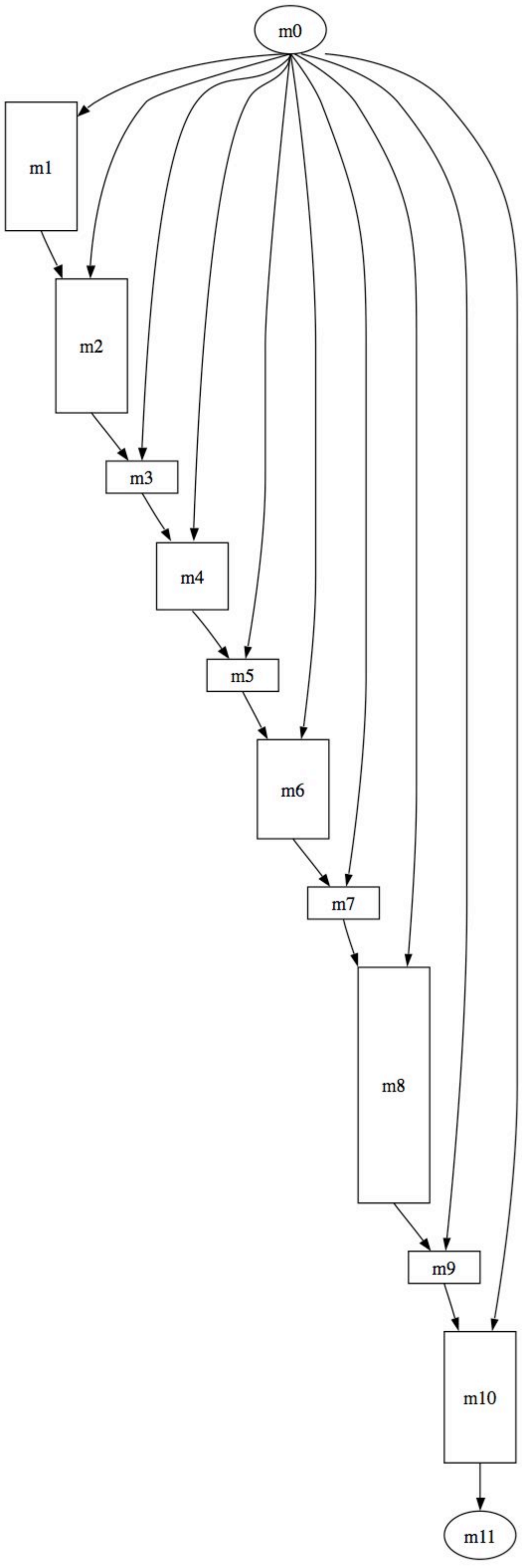
This is an example of a flat graph with the fanout and coupling interval increased dramatically to produce a short or squat graph. For flat graphs the coupling_period parameter corresponds to the reach of an output arc i.e. how many models it may span.



2 1 10 1.0 10 5 4 100 1 5.0 100000 10000 1

C. *Tall Ten Model Graph*

This is an example of a flat graph with the coupling interval and fanout reduced to produce a tall graph.



2 1 10 1.0 10 5 1 1 1 5.0 100000 10000 1

VII. Closing Comments

It is hoped that the Synthesiser outlined in this report will prove useful, or at least spark some enthusiasm, for further developments to support coupled-model research.

The Synthesiser is expected to be found at [7].

Acknowledgment

The authors thank the researchers in the Centre for Novel Computing at Manchester for the motivation to construct the Synthesiser.

REFERENCES

[1] Riley, G.D., 'Coupled-model Notation and Examples V1.0', Technical Report, Centre for Novel Computing, School of Computer Science, University of Manchester, 10 Nov 2006, *draft.*

[2] 'Bespoke Framework Generator Version 2 Specifications', Technical Report, Centre for Novel Computing, School of Computer Science, University of Manchester, *to follow*.

[3] Bane, K., et al., Armstrong, C., Ford, R., and Riley, G.D., 'A User Guide to Framework Generation using bfg', Technical Report, Centre for Novel Computing, School of Computer Science, University of Manchester, 2003.

[4] Ford, R.W., Riley, G.D., Bane, C.W., Armstrong, C.W., and Freeman, T.L., 'GCF: a general coupling framework', Concurrency and Computation: Practice and Experience' Volume 18, Issue 2 , pp 163 – 181, Special Issue: Computational Frameworks , Ed. Aad J. van der Steen. Published Online: 11 Oct 2005.

[5] Ford, R.W., and Riley, G.D., FLUME A *Fl*exible *U*nified *M*odel *E*nvironment Model Coupling Requirements, Met Office, 2002. *See [6].*

[6] Ford, R.W., and Riley, G.D., 'FLUME A *Fl*exible *U*nified *M*odel *E*nvironment D4 High Level Design', Met Office Project No. PB/A4872, Version 1.4, 10 Oct 2003, http://www.metoffice.gov.uk/research/interproj/flume/

[7] 'The Bespoke Framework Generator', http://www.cs.manchester.ac.uk/cnc/projects/bfg.php

[8] Graphviz, http://www.graphviz.org/

APPENDICES

m3.f

```
c     University of Manchester Centre for Novel Computing
c     Synthetic Workload Synthesiser G.K. Egan 2007
c     Generated  9 Mar 2007  13:37:50

       subroutine init_m3(init_m3_1)
      integer init_m3_1
c      empty
      return
      end

      subroutine end_m3()
c      empty
      return
      end

      subroutine m3(m3_1, m3_2, m3_3)
c       model interval 2
c       inputs  2
c       outputs 1
      integer i,j,k,mcheck
      integer a(58429),
     +    m3_1(6071),
     +    m3_2(4414),
     +
     +    m3_3(1384)

c      rudimentary signature generation
      mcheck=0
      mcheck=xor(mcheck, m3_1(1))
      mcheck=xor(mcheck, m3_2(1))
c time calls not used otherwise workload would vary
   with processor
      k=1
      do 100 i=1,580,1
        do 100 j=1,2150,1
c          run through the footprint sequentially modulo footprint
         k=mod(k+1, 58429)+1
        a(k)=a(k)+1
  100 continue

      return
      end
```

m3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- University of Manchester Centre for Novel Computing -->
<!-- Synthetic Workload Synthesiser G.K. Egan 2007 -->
<!-- Generated Models  9 Mar 2007  13:37:50    -->
<definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="BFG2HOME/bfg2/schema/
definition.xsd">
  <name>m3</name>
  <author>Synthesiser G.K. Egan</author>
  <type>scientific</type>
  <language>f77</language>
  <timestep>2</timestep>
   <entryPoints>
     <entryPoint type="init" name="init_m3">
        <data name="init_m3_1" direction="in" form="argpass"
              id="1" dataType="integer" dimension="0"></data>
     </entryPoint>
     <entryPoint type="iteration" name="m3">
       <data name="m3_1" direction="in" form="argpass"
                id="1" dataType="integer" dimension="1">
       <dim value="1">
         <lower><integer>1</integer></lower>
          <upper><integer>6071</integer></upper>
       </dim>
     </data>
       <data name="m3_2" direction="in" form="argpass"
                id="2" dataType="integer" dimension="1">
       <dim value="1">
         <lower><integer>1</integer></lower>
          <upper><integer>4414</integer></upper>
       </dim>
     </data>
       <data name="m3_3" direction="out" form="argpass"
                id="3" dataType="integer" dimension="1">
       <dim value="1">
         <lower><integer>1</integer></lower>
          <upper><integer>1384</integer></upper>
       </dim>
     </data>
     </entryPoint>
     <entryPoint type="final" name="end_m3">
     </entryPoint>
   </entryPoints>
</definition>
```

Compose.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- University of Manchester Centre for Novel Computing -->
<!-- Synthetic Workload Synthesiser G.K. Egan 2007 -->
<!-- Generated Composition  9 Mar 2007  13:37:50    -->
<composition
xmlns:xsi="http://www.w3.org/2001/xmlSchema-instance"
xsi:noNamespaceSchemaLocation="BFG2HOME/bfg2/schema/
composition.xsd">
  <connections>
    <timestepping>
     <set name="init_m1_1">
      <primed>
         <data dimension="0" datatype="integer" value="1"/>
      </primed>
        <field modelName="init_m1" epName="init_m1" id="1"/>
    </set>
     <set name="m1_2">
      <!-- integer(8416) -->
      <primed>
         <data dimension="1" datatype="integer" value="1"/>
      </primed>
       <field modelName="m1" epName="m1" id="2"/>
       <field modelName="m2" epName="m2" id="1"/>
    </set>
     <set name="init_m2_1">
      <primed>
         <data dimension="0" datatype="integer" value="2"/>
      </primed>
        <field modelName="init_m2" epName="init_m2" id="1"/>
    </set>
     <set name="m2_2">
      <!-- integer(8477) -->
      <primed>
         <data dimension="1" datatype="integer" value="2"/>
      </primed>
       <field modelName="m2" epName="m2" id="2"/>
      <field modelName="m1" epName="m1" id="1"/>
    </set>
    <set name="m2_3">
     <!-- integer(1812) -->
     <primed>
        <data dimension="1" datatype="integer" value="2"/>
     </primed>
      <field modelName="m2" epName="m2" id="3"/>
      <field modelName="m4" epName="m4" id="1"/>
    </set>
    <set name="init_m3_1">
     <primed>
        <data dimension="0" datatype="integer" value="3"/>
     </primed>
       <field modelName="init_m3" epName="init_m3" id="1"/>
    </set>
    <set name="m3_3">
     <!-- integer(1384) -->
     <primed>
        <data dimension="1" datatype="integer" value="3"/>
     </primed>
      <field modelName="m3" epName="m3" id="3"/>
      <field modelName="m4" epName="m4" id="2"/>
    </set>
    <set name="init_m4_1">
     <primed>
        <data dimension="0" datatype="integer" value="4"/>
     </primed>
       <field modelName="init_m4" epName="init_m4" id="1"/>
    </set>
    <set name="m4_3">
     <!-- integer(6071) -->
     <primed>
        <data dimension="1" datatype="integer" value="4"/>
     </primed>
      <field modelName="m4" epName="m4" id="3"/>
      <field modelName="m3" epName="m3" id="1"/>
    </set>
    <set name="m4_4">
     <!-- integer(4414) -->
     <primed>
        <data dimension="1" datatype="integer" value="4"/>
     </primed>
      <field modelName="m4" epName="m4" id="4"/>
      <field modelName="m3" epName="m3" id="2"/>
    </set>
   </timestepping>
  </connections>
</composition>
```

Coupling.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- University of Manchester Centre for Novel Computing -->
<!-- Synthetic Workload Synthesiser G.K. Egan 2007 -->
<!-- Generated Coupling  9 Mar 2007  13:37:50    -->
<coupled  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="BFG2HOME/bfg2/schema/
coupled.xsd">
  <models>
    <model>../m1/xml/m1.xml</model>
    <model>../m2/xml/m2.xml</model>
    <model>../m3/xml/m3.xml</model>
    <model>../m4/xml/m4.xml</model>
  </models>
   <composition>compose.xml</composition>
   <deployment>deploy.xml</deployment>
</coupled>
```

Deploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- University of Manchester Centre for Novel Computing -->
<!-- Synthetic Workload Synthesiser G.K. Egan 2007 -->
<!-- Generated Default Deploy  9 Mar 2007  13:37:50    -->
<deployment
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="BFG2HOME/bfg2/schema/
deployment.xsd">
  <deploymentUnits>
  <deploymentUnit language="f77">
   <sequenceUnit threads="1">
    <model name="m1"/>
   </sequenceUnit>
   <sequenceUnit threads="1">
    <model name="m2"/>
   </sequenceUnit>
   <sequenceUnit threads="1">
    <model name="m3"/>
   </sequenceUnit>
   <sequenceUnit threads="1">
    <model name="m4"/>
   </sequenceUnit>
  </deploymentUnit>
  </deploymentUnits>
  <target>mpi</target>
   <schedule>
    <bfgiterate>
     <init>
       <model name="m1" ep="init_m1"/>
       <model name="m2" ep="init_m2"/>
       <model name="m3" ep="init_m3"/>
       <model name="m4" ep="init_m4"/>
     </init>
     <iterate>
       <loop niters="8">
        <model name="m1" ep="m1"/>
        <model name="m2" ep="m2"/>
        <model name="m3" ep="m3"/>
        <model name="m4" ep="m4"/>
       </loop>
     </iterate>
     <final>
       <model name="m1" ep="end_m1"/>
       <model name="m2" ep="end_m2"/>
       <model name="m3" ep="end_m3"/>
       <model name="m4" ep="end_m4"/>
     </final>
    </bfgiterate>
   </schedule>
</deployment>
```

Mpi_m3.f

```
 c     University of Manchester Centre for Novel Computing
c     Synthetic Workload Synthesiser G.K. Egan 2007
c     Generated  9 Mar 2007  13:37:50

       subroutine mpi_m3(myid, iteration)
c     model interval 2
      include 'mpif.h'
       integer iteration, myid, istatus(mpi_status_size), ierr
     integer
    +    m3_1(6071),
    +    m3_2(4414),
    +
    +    m3_3(1384)
```

```
       if (iteration.le.2) then
         m3_1(1)=3
       else
         if (mod(iteration-2, 2).eq.0) then
           print *, iteration, "recv  m4->m3_1 [6071]"
            call mpi_recv(m3_1,6071,mpi_integer,
     +               4,1,mpi_comm_world,istatus,ierr)
         endif
       endif
       if (iteration.le.2) then
         m3_2(1)=3
       else
         if (mod(iteration-2, 2).eq.0) then
           print *, iteration, "recv  m4->m3_2 [4414]"
            call mpi_recv(m3_2,4414,mpi_integer,
     +               4,2,mpi_comm_world,istatus,ierr)
         endif
       endif

         print *, iteration, "exec  m3"
         call m3(m3_1, m3_2, m3_3)

         if (mod(iteration,2).eq.0) then
           print *, iteration, "send  m3_3->m4 [1384]"
            call mpi_send(m3_3,1384,mpi_integer,
     +              4,2,mpi_comm_world, ierr)
         endif

       return
       end
```

mpi_m.f

```
c     University of Manchester Centre for Novel Computing
c     Synthetic Workload Synthesiser G.K. Egan 2007
c     Generated  9 Mar 2007  13:37:50

       program mpi_m
       include 'mpif.h'
       integer iteration
c       integer init_m1_1, init_m2_1, init_m3_1, init_m4_1
c      call init_m1(init_m1)
c      call init_m2(init_m2)
c      call init_m3(init_m3)
c      call init_m4(init_m4)

       call mpi_init(ierr)
       call mpi_comm_rank(mpi_comm_world, myid, ierr)
       call mpi_comm_size(mpi_comm_world, numprocs, ierr)
       if (numprocs.lt.5) then
        call mpi_finalize(rc)
        print *, "number of processes must be 5"
        stop
       endif

c     major cycle length is 8
      do 100 iteration=1,80,1
        if ((mod(iteration,1).eq.0).and.(myid.eq.1)) then
          call mpi_m1(myid, iteration)
        endif
        if ((mod(iteration,4).eq.0).and.(myid.eq.2)) then
          call mpi_m2(myid, iteration)
        endif
        if ((mod(iteration,2).eq.0).and.(myid.eq.3)) then
          call mpi_m3(myid, iteration)
        endif
        if ((mod(iteration,2).eq.0).and.(myid.eq.4)) then
          call mpi_m4(myid, iteration)
        endif
        call mpi_barrier(mpi_comm_world, ierr)
  100 continue
      call mpi_finalize(rc)

c     call end_m1()
c     call end_m2()
c     call end_m3()
c     call end_m4()

      stop
      end
```

model.dag

```
% University of Manchester Centre for Novel Computing
% Synthetic Workload Synthesiser G.K. Egan 2007
% Generated  9 Mar 2007  13:37:50
%   the node id is computed as (iteration-1)*Models+baseid
    where Models in this case=4
%   because of coupling the unrolled node id space is sparse
%   the actual number of instances of models
     unrolled including the null source and sink is:
20
%   connections and weights (node node weight (~seconds))
1 5 0
5 9 0
9 13 0
13 17 0
17 21 0
21 25 0
25 29 0
29 33 0
14 30 0
30 33 0
7 15 0
……….
14 24 1812
15 24 1384
16 23 6071
16 23 4414
23 33 1384
24 33 6071
24 33 4414
```

model.cost

```
% University of Manchester Centre for Novel Computing
% Synthetic Workload Synthesiser G.K. Egan 2007
% Generated  9 Mar 2007  13:37:50
% active ids and no of processors
20 4
% cost of task i on mach j (~seconds)
      0  0.000e+00 0.000e+00 0.000e+00 0.000e+00
      1  1.024e+00  1.024e+00  1.024e+00  1.024e+00
      5  1.024e+00  1.024e+00  1.024e+00  1.024e+00
…..
     29  1.024e+00  1.024e+00  1.024e+00  1.024e+00
     30  3.284e+00  3.284e+00  3.284e+00  3.284e+00
     31  5.800e-01  5.800e-01  5.800e-01  5.800e-01
     32  4.911e+00  4.911e+00  4.911e+00  4.911e+00
     33  0.000e+00  0.000e+00  0.000e+00  0.000e+00
```

A BFG2 coupled model 20 models and a major cycle of 120
0 1 20 1.0 20 5 2 5 1 5.0 100000 10000 1