Joint Structured/ Unstructured Parallelism Exploitation in muskel

M. Danelutto * ^ & P. Dazzi ° " ^
* Dept. Computer Science - Univ. Pisa
° ISTI/CNR - Pisa
" Institute for Advanced Studies - Lucca
^ CoreGRID Institute on Programmig Model

Outline

- Skeletons / muskel
- Macro data flow implementation
- Joint parallelism exploitation
- Experimental results
- Conclusions

Outline

- Skeletons / muskel
- Macro data flow implementation
- Joint parallelism exploitation
- Experimental results
- Conclusions

Skeletons

- Useful, parametric, efficient parallelism exploitation pattern
  - **useful** : for a large class of applications
  - **parametric** : in the seq code, parallelism degree, types of tasks and results
  - **efficient** : known efficient implementations on a range of architectures

Sample ske: farm

- **farm: ('a → 'b) → stream 'a → stream 'b** possibly all workers:('a → 'b) are computed in parallel
- **useful** most of currently large scale parallel application fit the schema
- **parametric** in the code, the data types and in parallelism degree
- **efficient** master slave, SMP multithread, ...

Typical skeletal sys

- stream parallel skeletons
  pipeline, farm, while, repeat, ...
- data parallel skeletons
  map, reduce, prefix, ...
- sequential code skeletons
  seq code wrapped into skeletons
- nesting
  free, limited (two tier), non allowed at all, ...

Typical skeletal sys (2)

- usually implemented with process template tecnology
  - P3L (UNIPI '91, MPI) -> SkIE ('96), Assist (Vanneschi '01, TCP/IP sock), Muesli (Kuchen '01, MPI), eSkel (Cole '02, MPI)

Pros

- Programmers
  - pick up a skeleton (composition)
  - provide code parameters
  - compile/run
- Skeletal system
  - provide efficient, correct and safe implementation, with optimizations!

Cons

- fixed skeleton set ➩ not possible to exploit (even slightly) different patterns
- poor / no interoperability with other parallel frameworks
- constrains in nesting (two tier)
- run time / libraries needed to run object code programs

muskel

- full Java skeleton library
- stream parallel skeleton subset
  (at the moment)
- derived from Lithium (Danelutto,Teti, 2000)
- fully nestable skeletons
- exploits Macro Data Flow (MDF) technology

muskel program

```
import muskel.*;

public class SampleCode {

  public static void main(String [] a) {

    Compute inc1 = new Inc();
    Compute sq1 = new Square();
    Compute f1 = new Farm(sq1);
    Compute main =
        new Pipeline(inc1,f1);

    Manager mgr = new Manager(main,"in.dat","out.dat");
    mgr.setContract(new ParDegree(10));
    mgr.compute();
  }

}
```

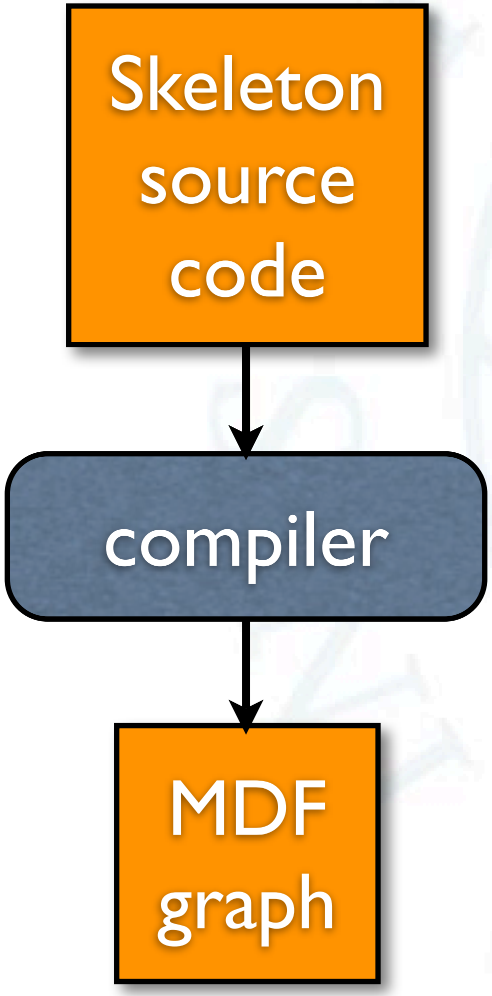
Outline

- Skeletons / muskel
- Macro data flow implementation
- Joint parallelism exploitation
- Experimental results
- Conclusions

Macro Data Flow

- macro data flow (Parco'99 -> PPL'00)
  - Lithium ('00) → Skipper ('01) → muskel ('04)
  - alternative implementation model (w.r.t. process templates)
- Step 1: translate skeleton tree into (macro) data flow graphs
- Step 2: instantiate 1 graph per input task and execute it on a distributed MDF interpreter

Step 1: compile




```
Compute inc1 = new Inc();
Con
Con
Con
```

```
public static MdfGraph
    compile(Compute pgm,CodeStorage cs) {
  if( pgm instanceof Farm) {
    Compute worker =  ((Farm) pgm).getWorker();
    MdfGraph w = compile(worker,cs);
    return w;}
  if(pgm instanceof Pipeline) {
}
```

```
GRAPH is:
instr:0:MDFI
<1,null>:op=18:in=1:out=1:fireable=false:
  tokens:<null,false>:destTokens:<null,0,2,9>
instr:1:MDFI
<2,null>:op=19:in=1:out=1:fireable=false:
  tokens:<null,false>:destTokens:<null,0,-1,9>
```

Step 1: compile (2)

- instanceof Farm -> compile(worker)
- instanceof Pipeline ->
  compile(stage1)
  compile(stage2)
  reloc(stage2)
  redirect ouput(stage1) to input(stage2)
- instanceof Compute (seq) ->
  1 input 1 output (unbound) token MDFi

Step 2: instantiate

```
GRAPH is:
instr:0:MDFI
<1,null>:op=18:in=1:out=1:fireable=false:
  tokens:<null,false>:destTokens:<null,0,2,9>
instr:1:MDFI
<2,null>:op=19:in=1:out=1:fireable=false:
  tokens:<null,false>:destTokens:<null,0,-1,9>
```

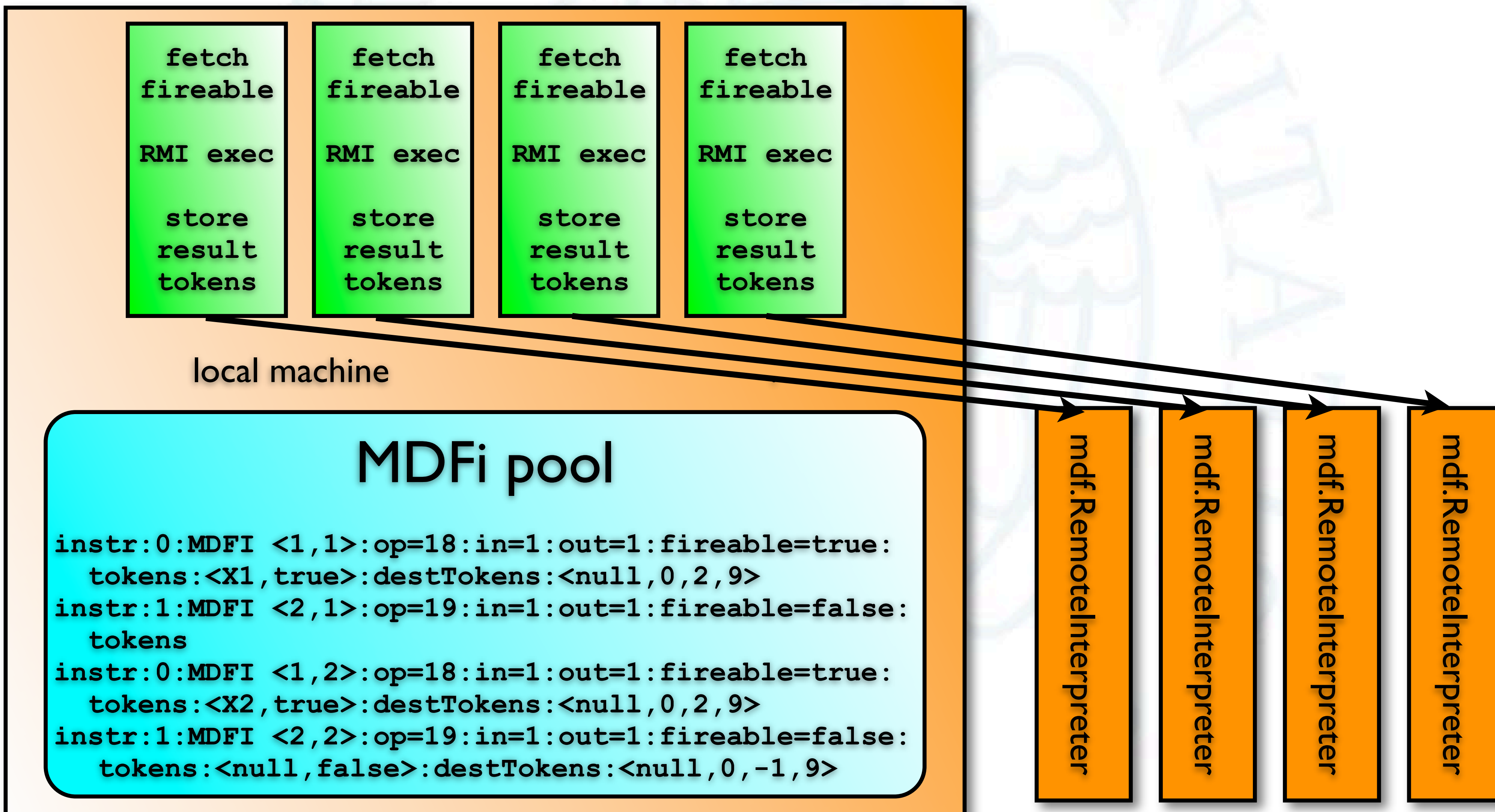
```
token X1 on input stream
token X2 on input stream
token X3 on input stream
```

```
GRAPH is:
instr:(
<1,1>:(
  toker
instr:1
<2,1>:(
  toker
```

```
GRAPH is:
instr:(
<1,2>:(
  toker
instr:1
<2,2>:(
  toker
```

```
GRAPH is:
instr:0:MDFI
<1,3>:op=18:in=1:out=1:fireable=true:
  tokens:<X3,true>:destTokens:<null,0,2,9>
instr:1:MDFI
<2,3>:op=19:in=1:out=1:fireable=false:
  tokens:<null,false>:destTokens:<null,0,-1,9>
```

Step 2: interpreter



Outline

- Skeletons / muskel
- Macro data flow implementation
- Joint parallelism exploitation
- Experimental results
- Conclusions

Cole's requirements

I. propagate the concept with minimal disruption

II. integrate ad hoc parallelism

III. accommodate diversity

IV. show the pay-back

Cole's requirements

I. propagate the concept with minimal disruption

II. integrate ad hoc parallelism

III. accommodate diversity

IV. show the pay-back

Joint par exploitation

- Provides users the possibility to write complete MDF graphs
  - modeling parallelism exploitation patterns not covered by standard skeletons
- Provides suitable ways to embed user defined MDF graphs into/with skeleton code
- Provides efficient implementation
- Fundamental to meet Cole's requirements

User def MDF

- ParCompute class to wrap MDF graphs into skeletons
- utilities to support creation of MDF instructions
- and to insert MDF instructions into MDF graphs

User def MDF (2)

```
Compute inc1 = new Inc();
Dest d = new Dest(0, 2 ,Mdfi.NoGraphId);
Dest[] dests = new Dest[1];
dests[0] = d;
Mdfi i1 = new Mdfi(manager,1,inc1,1,1,dests);

Compute sq1 = new Square();
Dest d1 = new Dest(0,Mdfi.NoInstrId, Mdfi.NoGraphId);
Dest[] dests1 = new Dest[1];
dests1[0] = d1;
Mdfi i2 = new Mdfi(manager,2,sq1,1,1,dests1);

MdfGraph graph = new MdfGraph();
graph.addInstruction(i1);
graph.addInstruction(i2);

ParCompute userDefMDFg = new ParCompute(graph);
```

Skeleton embedding

- ParCompute used in all places where a Compute module can be used

```
Compute sq =
    new Square();
Farm s2 =
    new Farm(sq);
Pipeline main = new
    Pipeline(userDefMDFg,s2);
```

- that means:
  - skeletons with arbitrary stager/workers
  - programs with skeletons, user defined MDF graph, combination of the two

Efficiency

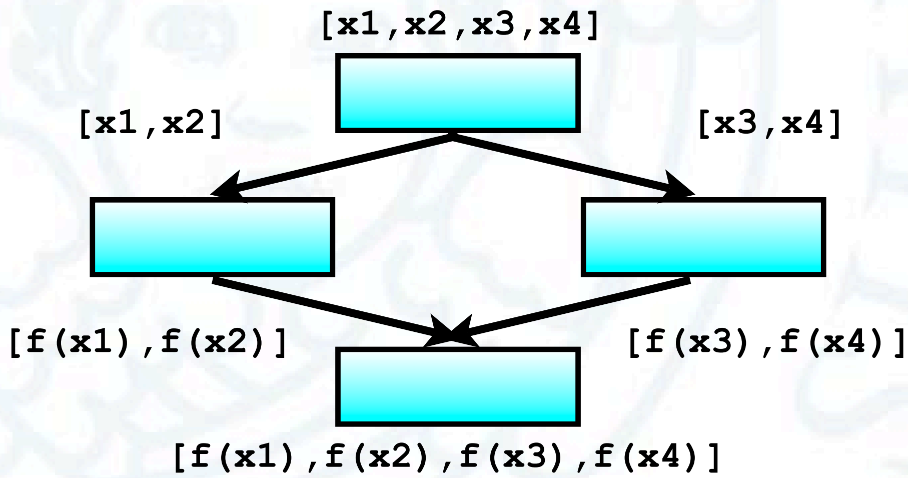
- Distributed MDF interpreter
  - processes MDF instructions "compiled" from skeleton code
  - as well as those provided by user
- Same efficiency provided computation grain is decent

Skel expandability

- User may develop "once&forall" new useful skeletons
- Embed them into proper Compute subclasses
- Use them again and again in further applications
- Provide them to "community users"

New skel sample

- Map2: get a vector, split in two, apply a worker on the two halves and recompose vector result



- not present in base muskel …

The code ...

```
public class Map2 extends ParCompute {
   public Map2(Compute f, Manager manager) {
            program = new MdfGraph();
      Dest [] dds1 = new Dest[2];
      dds1[0]=new Dest(0,2);
      dds1[1]=new Dest(0,3);
      Mdfi emitter = new Mdfi(manager,1,new MapEmitter(2),1,2,dds1);
      program.addInstruction(emitter);
      Dest [] dds2 = new Dest[1];
      dds2[0] = new Dest(0,4);
      Mdfi if1 = new Mdfi(manager,2, f, 1, 1, dds2);
      program.addInstruction(if1);
      Dest []dds3 = new Dest[1];
      dds3[0] = new Dest(1,4);
      Mdfi if2 = new Mdfi(manager,3, f, 1, 1, dds3);
      program.addInstruction(if2);
      Dest[] ddslast = new Dest[1];
      ddslast[0] = new Dest(0,Mdfi.NoInstrId);
      Mdfi coll = new Mdfi(manager,4,new MapCollector(),2,1,ddslast);
      program.addInstruction(collector);
      return;
   }
```

The program ...

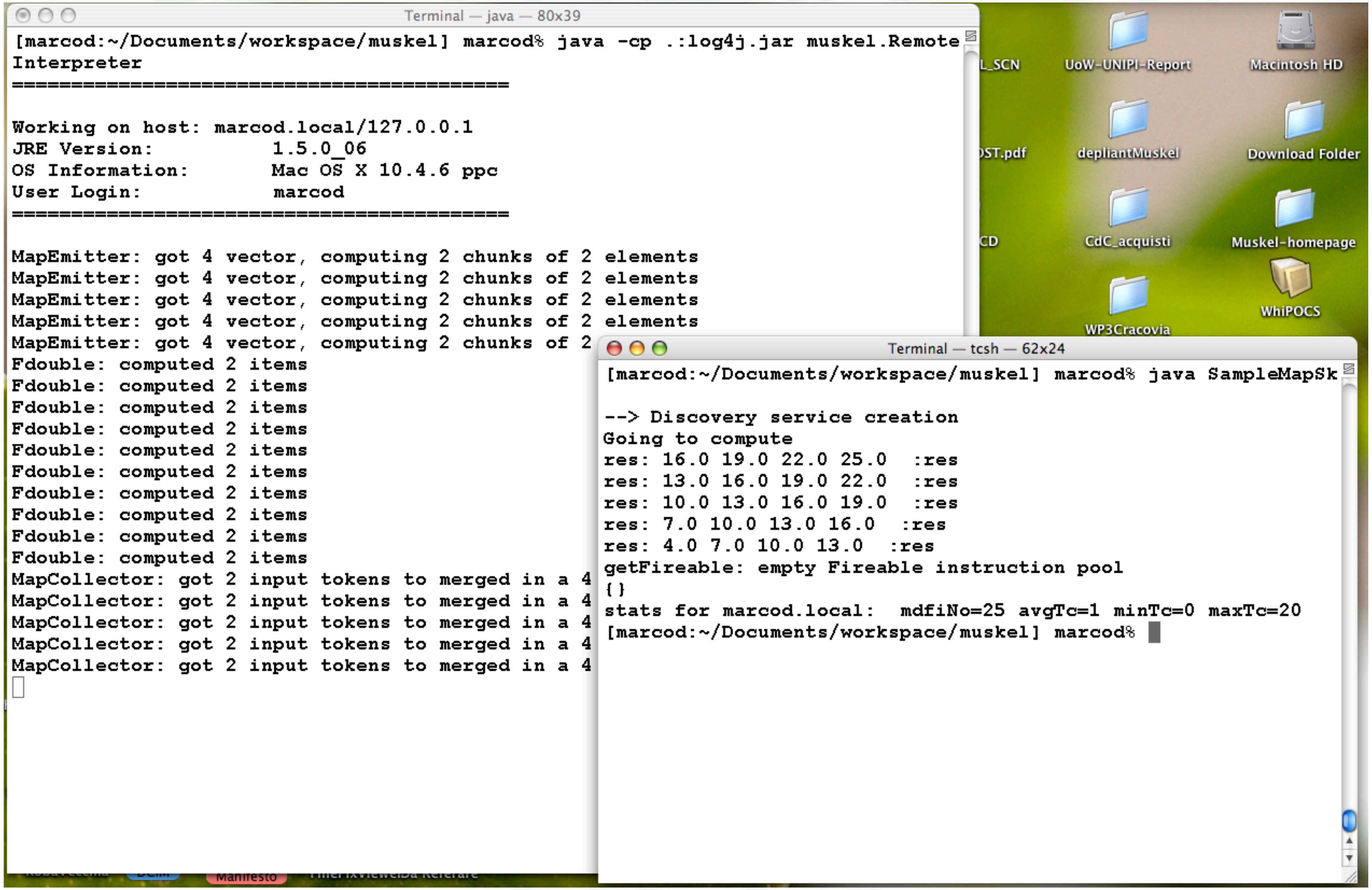
```
public static void main(String[] args) {
    Manager manager = new Manager();

    Compute seqStage = new IncDoubleVector();
    Compute worker   = new Fdouble();
    Compute mapStage = new Map2(worker,manager);
    Pipeline main = new Pipeline(mapStage,seqStage);

    InputManager inManager = new DoubleVectIM(5,4); // 5 tasks (#=4)
    OutputManager outManager = new DoubleVectOM();
    ParDegree contract = new ParDegree(Integer.parseInt(args[0]);
    manager.setInputManager(inManager);
    manager.setOutputManager(outManager);
    manager.setContract(contract);
    manager.setProgram(main);

    manager.compute();
}
```

The run ...



Terminal — java — 80x39
[marcod:~/Documents/workspace/muskel] marcod% java -cp .:log4j.jar muskel.Remote
Interpreter
==
Working on host: marcod.local/127.0.0.1
JRE Version: 1.5.0_06
OS Information: Mac OS X 10.4.6 ppc
User Login: marcod
==
MapEmitter: got 4 vector, computing 2 chunks of 2 elements
MapEmitter: got 4 vector, computing 2 chunks of 2 elements
MapEmitter: got 4 vector, computing 2 chunks of 2 elements
MapEmitter: got 4 vector, computing 2 chunks of 2 elements
MapEmitter: got 4 vector, computing 2 chunks of 2
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
Fdouble: computed 2 items
MapCollector: got 2 input tokens to merged in a 4
MapCollector: got 2 input tokens to merged in a 4
MapCollector: got 2 input tokens to merged in a 4
MapCollector: got 2 input tokens to merged in a 4
MapCollector: got 2 input tokens to merged in a 4
Terminal — tcsh — 62x24
[marcod:~/Documents/workspace/muskel] marcod% java SampleMapSk
--> Discovery service creation
Going to compute
res: 16.0 19.0 22.0 25.0 :res
res: 13.0 16.0 19.0 22.0 :res
res: 10.0 13.0 16.0 19.0 :res
res: 7.0 10.0 13.0 16.0 :res
res: 4.0 7.0 10.0 13.0 :res
getFireable: empty Fireable instruction pool
{}
stats for marcod.local: mdfiNo=25 avgTc=1 minTc=0 maxTc=20
[marcod:~/Documents/workspace/muskel] marcod%
UoW-UNIPI-Report
Macintosh HD
depliantMuskel
Download Folder
CdC_acquisti
Muskel-homepage
WP3Cracovia
WhiPOCS

Outline

- Skeletons / muskel
- Macro data flow implementation
- Joint parallelism exploitation
- Experimental results
- Conclusions

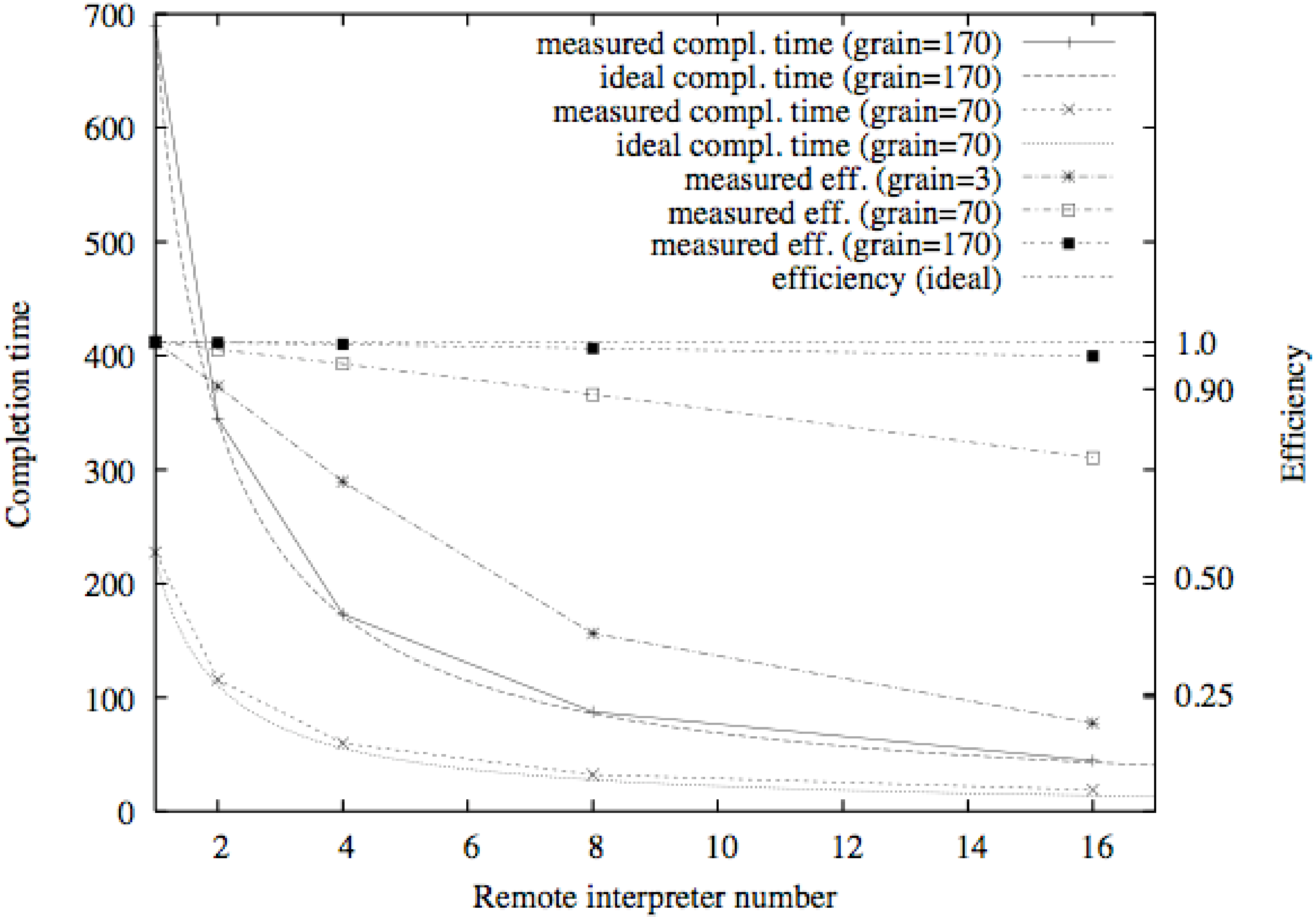
Experiments

1. Plain efficiency / scalability
2. Load balancing experiment with mixed user defined MDF and Skeleton code
3. Heterogeneous architecture experiment with Linux-Pentium and MacOS/X-PowerPC RemoteInterpreters

Efficiency

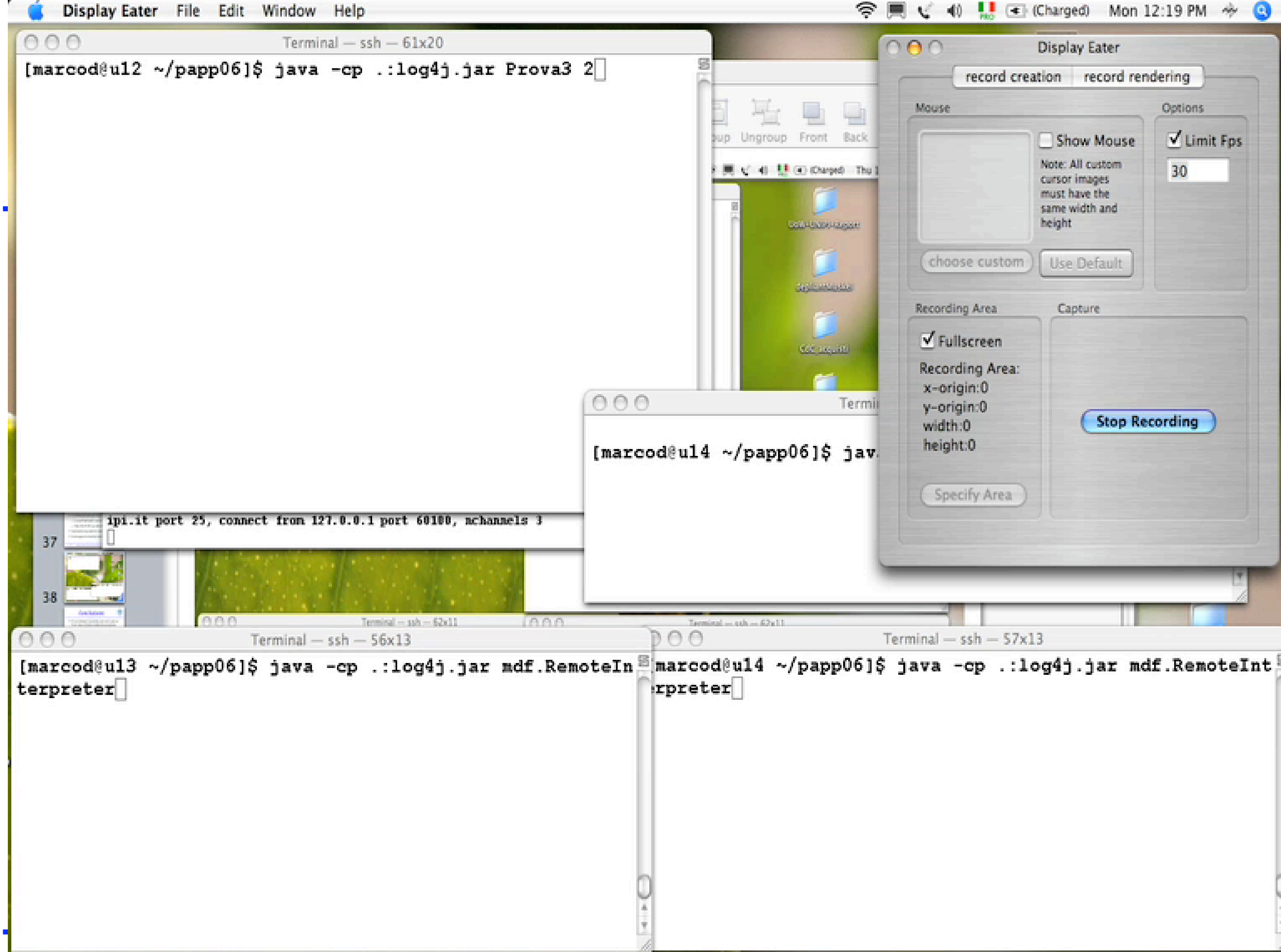
- Sample code with syntetic MDF instructions
- Run on a variable number of muskel.RemoteInterpreter process instances
- Scalability and efficiency measured
- Gives a precise idea of the "suitable" grains that can be exploited

Efficiency (2)



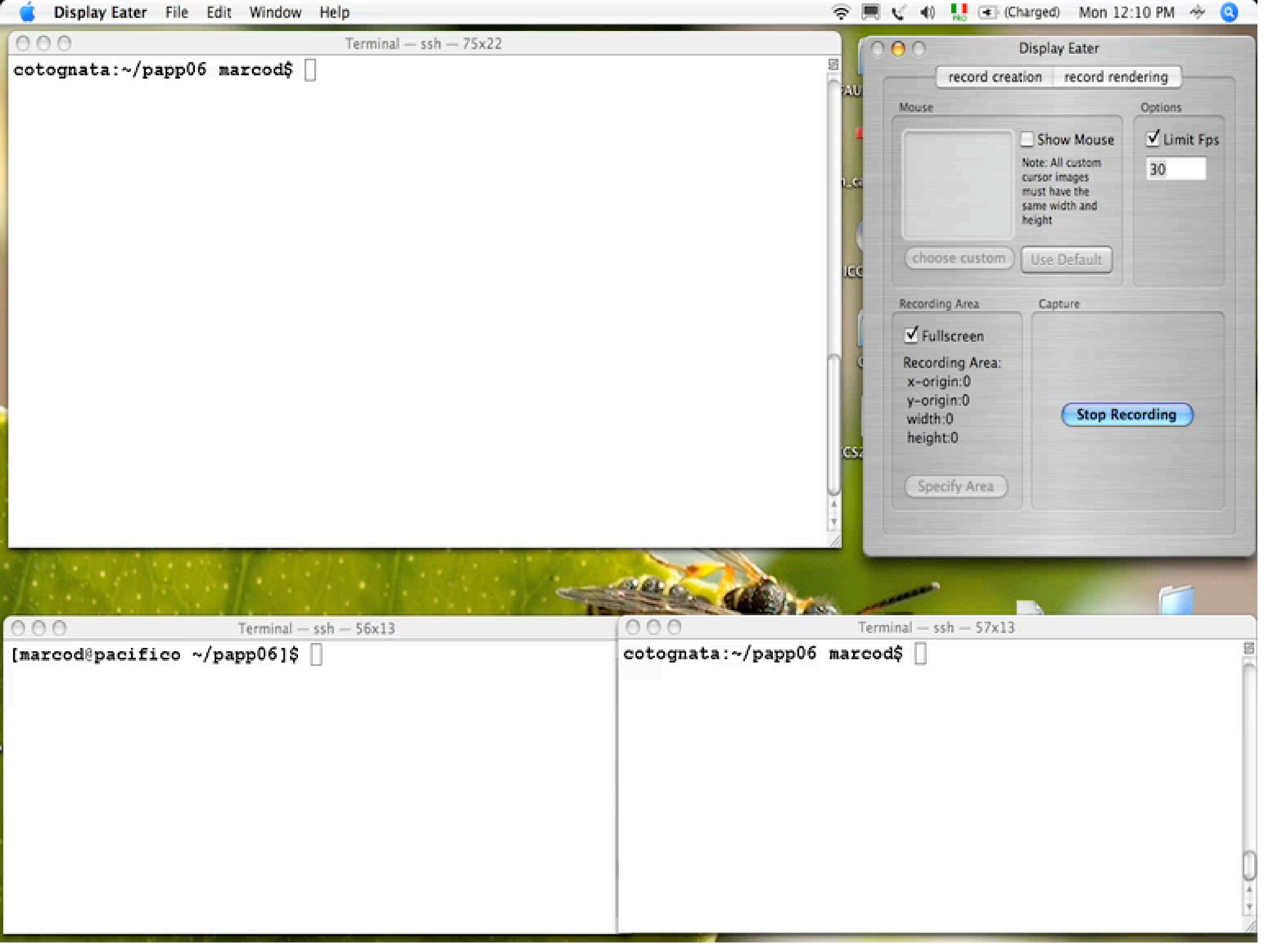
Load balancing

- 2 mdf.RemoteInterpreters
  - on a Linux/PentiumIII RLX blade cluster
- Run 1
  - 1/2 MDFi on first RemoteInterpeter
  - 1/2 MDFi on the second one
- Run 2 : additional load on one machine
  - more MDFi on the second one


Display Eater File Edit Window Help
(Charged) Mon 12:19 PM
Terminal — ssh — 61x20
[marcod@u12 ~/papp06]$ java -cp .:log4j.jar Prova3 2
Display Eater
record creation
record rendering
Mouse
Options
Show Mouse
Limit Fps
30
Note: All custom cursor images must have the same width and height
choose custom
Use Default
Recording Area
Capture
Fullscreen
Recording Area:
x-origin:0
y-origin:0
width:0
height:0
Stop Recording
Specify Area
Ungroup
Front
Back
[marcod@u14 ~/papp06]$ jav
ipi.it port 25, connect from 127.0.0.1 port 60100, nchannels 3
37
38
Terminal — ssh — 56x13
[marcod@u13 ~/papp06]$ java -cp .:log4j.jar mdf.RemoteIn
terpreter
Terminal — ssh — 57x13
marcod@u14 ~/papp06]$ java -cp .:log4j.jar mdf.RemoteInt
rpreter

Heterogeneous

- 2 mdf.RemoteIntepreters
  - 1 Linux/PentiumIV server (remote)
  - 1 MacOS/X PPC workstation (local)
- load balancing exploits local machine
- heterogeneity handled natively by Java


Display Eater File Edit Window Help
(Charged) Mon 12:10 PM
Terminal — ssh — 75x22
cotognata:~/papp06 marcod$
Display Eater
record creation
record rendering
Mouse
Show Mouse
Note: All custom cursor images must have the same width and height
choose custom
Use Default
Options
Limit Fps
30
Recording Area
Fullscreen
Recording Area:
x-origin:0
y-origin:0
width:0
height:0
Specify Area
Capture
Stop Recording
Terminal — ssh — 56x13
[marcod@pacifico ~/papp06]$
Terminal — ssh — 57x13
cotognata:~/papp06 marcod$

Conclusions

- First attempt to provide user def code as first class citizien in skeleton programs
- Mechanism to be provided to programmers still under development
- Very encouraging experimental results
- Current work: extension to allow dynamic MDF graph generation (support recursion and dynamic skeletons)

Conclusions (2)

- version of muskel on top of ProActive (INRIA/OASIS) will run (soon) on top of workstation networks, clusters and GRIDS
- muskel available GPL (new release May06!) http://www.di.unipi.it/~marcod/Muskel
- Ask questions to marcod@di.unipi.it

Thank you for the attention

any questions ?