Running Csound in Parallel
LAC2009

John ffitch

University of Bath

LAC Maynooth, May 2011
Introduction

We are seeing more processors rather than faster ones.

The challenge now is to find ways of using multiple cores effectively to improve the performance of a single program.

NB: No interest here in efficiency
I first stated this in the early 1970s, and at intervals since, but now the need is much more imminent!
Robert S. Barton, one of the greats of early computing, said there is a *technological imperative*; what hardware requires is a forcing term on software.

Attempts at parallelism at that time did not succeed, but we can learn from the experience.
Robert S. Barton, one of the greats of early computing, said there is a *technological imperative*; what hardware requires is a forcing term on software.

Attempts at parallelism at that time did not succeed, but we can learn from the experience.
Forty years ago I was proposing a parallel functional machine, and thirty years ago we built the Bath Concurrent LISP Machine, a cluster of six M68000 processors with each processor having three shared memory windows with one other.

Twenty years ago we built the LISP-based Concurrent Object-Oriented system.
 Forty years ago I was proposing a parallel functional machine, and thirty years ago we built to Bath Concurrent LISP Machine, a cluster of six M68000 processors with each processor having three shared memory windows with one other.

Twenty years ago we built the a LISP-based Concurrent Object-Oriented system.
We based our work on the premise that users cannot be expected (or trusted) to modify their thinking for parallel execution, and the responsibility needs to be taken by the software translation system that converts the program or specification into an executable form.

Compiler analysis can be extended to inform the structure; described variously in PhD Thesis of Marti (1980), and papers by me in computer algebra.
Concurrent Software

We based our work on the premise that users cannot be expected (or trusted) to modify their thinking for parallel execution, and the responsibility needs to be taken by the software translation system that converts the program or specification into an executable form.

Compiler analysis can be extended to inform the structure; described variously in PhD Thesis of Marti (1980), and papers by me in computer algebra.
The Two Critical Points

A: Two entities can be run at the same time if they do not reference/modify shared data

B: Two entities should be run at the same time if the overhead is less than the gain
The Two Critical Points

A: Two entities can be run at the same time if they do not reference/modify shared data.

B: Two entities should be run at the same time if the overhead is less than the gain.
Should we just start coding again? I say not as there is too much already committed.

Two attempts however are worth mentioning;
- Csound in real-time using Transputers
- Midas streamed DSP network

Both are finer-grained than what I am advocating
Should we just start coding again? I say not as there is too much already committed.

Two attempts however are worth mentioning;
- Csound in real-time using Transputers
- Midas streamed DSP network
Both are finer-grained than what I am advocating.
Towards a Parallel Csound

Csound has been in existence and development for 25 years. It provides **instruments** that are played following a score.

The instruments are activated, performed and deactivated using a control cycle (running at a control rate). Instruments are performed in a defined order, and so interaction between instruments has defined behaviour.

```
until end of events do
deal with notes ending
sort new events onto instance list
for each instrument in instance list
  calculate instrument
```
Towards a Parallel Csound

Csound has been in existence and development for 25 years. It provides **instruments** that are played following a score.

The instruments are activated, performed and deactivated using a control cycle (running at a control rate). Instruments are performed in a defined order, and so interaction between instruments has defined behaviour.

```plaintext
until end of events do
  deal with notes ending
  sort new events onto instance list
  for each instrument in instance list
    calculate instrument
```
Towards a Parallel Csound (b)

Making this parallel could be to make the loop parallel, as long as there is no interaction, so....

- Following Marti we can use code analysis techniques.
- Only global variables matter.
- For each instrument determine the sets of globals are read, written, or both.
- Use this to control the loop.
Towards a Parallel Csound (b)

Making this parallel could be to make the loop parallel, as long as there is no interaction, so....

- Following Marti we can use code analysis techniques
- Only global variables matter.
- For each instrument determine the sets of globals are read, written, or both
- Use this to control the loop
Towards a Parallel Csound (b)

Making this parallel could be to make the loop parallel, as long as there is no interaction, so....

- Following Marti we can use code analysis techniques
- Only global variables matter.
- For each instrument determine the sets of globals are read, written, or both
- Use this to control the loop
Towards a Parallel Csound (b)

Making this parallel could be to make the loop parallel, as long as there is no interaction, so....

- Following Marti we can use code analysis techniques
- Only global variables matter.
- For each instrument determine the sets of globals are read, written, or both
- Use this to control the loop
Towards a Parallel Csound (b)

Making this parallel could be to make the loop parallel, as long as there is no interaction, so....

- Following Marti we can use code analysis techniques
- Only global variables matter.
- For each instrument determine the sets of globals are read, written, or both
- Use this to control the loop
Special case: most instruments add into the output bus, but this is not an operation that needs ordering (subject to rounding errors), although it may need a mutex or spin-lock. The language processing can insert any necessary protections in these cases.

There are other globals than variables but the idea is the same.
Towards a Parallel Csound (c)

Special case: most instruments add into the output bus, but this is not an operation that needs ordering (subject to rounding errors), although it may need a mutex or spin-lock. The language processing can insert any necessary protections in these cases.

There are other globals than variables but the idea is the same.
Design

Build a DAG of ordering dependency, where the arcs represent the need to be evaluated before the descendents

until end of events do
    deal with notes ending
    add new events and reconstruct the DAG
until DAG empty
    foreach processor
        evaluate a root from DAG
    wait until all processes finish
Design

Build a DAG of ordering dependency, where the arcs represent the need to be evaluated before the descendents

until end of events do
    deal with notes ending
    add new events and reconstruct the DAG
until DAG empty
    foreach processor
        evaluate a root from DAG
    wait until all processes finish
Using the new parser the information is “easily” gathered and the bus-locks inserted.

```
instr 1
    a1 oscil p4, p5, 1
    out a1
endin
instr 2
    gk oscil p4, p5, 1
endin
instr 3
    a1 oscil gk, p5, 1
    out a1
endin
```
Instr1: [r:{ }; w:{ }; easy]
Instr2: [r:{ }; w:{gk}; easy]
Instr3: [r:{gk}; w:{ }; easy]
This is a major problem. It is consumed on each cycle, but adding and losing instances means DAG must be remade, not just copied. The current version of representation and algorithm is the result of much experimentation and probably could be improved.
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
We use the POSIX pthreads library.

- One master thread does analysis and DAG construction
- A Barrier at the start of each control cycle
- Each worker gets a task from the DAG, with a mutex
- At end of instrument-cycle DAG is modified
- When no work proceed to end Barrier
Load Balancing

We would like each task to be equal computation and sufficiently large.

This is not always true and currently we ignore this problem.

Code exists to collect instances together.
We would like each task to be equal computation and sufficiently large.

This is not always true and currently we ignore this problem.

Code exists to collect instances together.
We would like each task to be equal computation and sufficiently large.

This is not always true and currently we ignore this problem.

Code exists to collect instances together.
We are collecting data on average instruction count for opcodes, using valgrind.

We calculate three counts; initialisation, per k-cycle, per sample.
## Costs of a few opcodes

<table>
<thead>
<tr>
<th>Opcode</th>
<th>init</th>
<th>Audio</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>table.a</td>
<td>93</td>
<td>23.063</td>
<td>43.998</td>
</tr>
<tr>
<td>table.k</td>
<td>93</td>
<td>0</td>
<td>45</td>
</tr>
<tr>
<td>butterlp</td>
<td>9</td>
<td>29.005</td>
<td>5.478</td>
</tr>
<tr>
<td>butterhi</td>
<td>19</td>
<td>30.000</td>
<td>35</td>
</tr>
<tr>
<td>butterbp</td>
<td>20</td>
<td>30</td>
<td>71</td>
</tr>
<tr>
<td>bilbar</td>
<td>371.5</td>
<td>1856.028</td>
<td>86</td>
</tr>
<tr>
<td>ags</td>
<td>497</td>
<td>917.921</td>
<td>79475.155</td>
</tr>
<tr>
<td>oscil.kk</td>
<td>69</td>
<td>12</td>
<td>47</td>
</tr>
<tr>
<td>oscili.kk</td>
<td>69</td>
<td>21</td>
<td>49</td>
</tr>
<tr>
<td>reverb</td>
<td>6963.5</td>
<td>77</td>
<td>158</td>
</tr>
</tbody>
</table>
Implemented by Chris Wilson, revised by John ffitch. Tested on Linux (and OSX). Requires the new parser but is available on Sourceforge as a branch. Can control number of threads.

Some features still missing, like zak and buses.
Implmented by Chris Wilson, revised by John ffitch. Tested on Linux (and OSX). Requires the new parser but is available on Sourceforge as a branch. Can control number of threads.

Some features still missing, like zak and buses.
## Linux Quadcore Results

<table>
<thead>
<tr>
<th>Sound</th>
<th>ksmps</th>
<th>1</th>
<th>5</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xanadu</td>
<td>1</td>
<td>31.202</td>
<td>39.291</td>
<td>42.318</td>
</tr>
<tr>
<td>Xanadu</td>
<td>100</td>
<td>16.023</td>
<td>17.413</td>
<td>16.999</td>
</tr>
<tr>
<td>Xanadu</td>
<td>300</td>
<td>17.159</td>
<td>16.137</td>
<td>15.141</td>
</tr>
<tr>
<td>CloudStrata</td>
<td>1</td>
<td>173.757</td>
<td>191.421</td>
<td>211.295</td>
</tr>
<tr>
<td>CloudStrata</td>
<td>10</td>
<td>89.406</td>
<td>80.998</td>
<td>94.023</td>
</tr>
<tr>
<td>CloudStrata</td>
<td>100</td>
<td>85.966</td>
<td>86.114</td>
<td>81.909</td>
</tr>
<tr>
<td>CloudStrata</td>
<td>300</td>
<td>87.153</td>
<td>76.045</td>
<td>79.353</td>
</tr>
<tr>
<td>CloudStrata</td>
<td>900</td>
<td>82.612</td>
<td>76.434</td>
<td>64.368</td>
</tr>
<tr>
<td>trapped</td>
<td>1</td>
<td>20.931</td>
<td>63.492</td>
<td>81.654</td>
</tr>
<tr>
<td>trapped</td>
<td>100</td>
<td>1.388</td>
<td>1.810</td>
<td>1.928</td>
</tr>
<tr>
<td>trapped</td>
<td>300</td>
<td>1.319</td>
<td>1.181</td>
<td>1.205</td>
</tr>
<tr>
<td>trapped</td>
<td>900</td>
<td>1.236</td>
<td>1.025</td>
<td>1.085</td>
</tr>
</tbody>
</table>
As the control rate decreases, corresponding to an increase in ksmps, the potential gain increases. This suggests that the current system is using too small a granularity and the collecting of instruments into larger groups will give a performance gain.

The performance figures are perhaps a little disappointing, but they do show that it is possible to get speed improvements, and more work on the load balance could be useful.
As the control rate decreases, corresponding to an increase in ksmps, the potential gain increases. This suggests that the current system is using too small a granularity and the collecting of instruments into larger groups will give a performance gain.

The performance figures are perhaps a little disappointing, but they do show that it is possible to get speed improvements, and more work on the load balance could be useful.
A system for parallel execution of Csound has been presented, that works at the granularity of the instrument, based on thirty-year old technology.

I believe that the level of granularity is the correct one, and with more attention to the DAG construction and load balancing it offers real gains for many users. It does not require specialist hardware, and can make use of current and projected commodity systems.
A system for parallel execution of Csound has been presented, that works at the granularity of the instrument, based on thirty-year old technology.

I believe that the level of granularity is the correct one, and with more attention to the DAG construction and load balancing it offers real gains for many users. It does not require specialist hardware, and can make use of current and projected commodity systems.
Acknowledgements

- Jed Marti (ex U of Utah and RAND; ARTIS, LLC)
- Arthur Norman (Trinity Collge, Cambridge)
- Chris Wilson (ex U of Bath; Imagination Technology plc)
- Steven Yi
- Csound Community

*Thanks to Codemist Ltd. This work in unsupported by public agencies*