AUTOMATIC PARALLELISATION OF SOFTWARE USING GENETIC IMPROVEMENT

Bobby R. Bruce
INSPIRATION

Samsung Galaxy S7

BOBBY R. BRUCE
INSPIRATION

Samsung Galaxy S7

Mali-T880 MP12
INSPIRATION

Samsung Galaxy S7

Mali-T880 MP12

Intel i7-2500K (overclocked to 5GHz)
INSPIRATION

Samsung Galaxy S7

Mali-T880 MP12

Intel i7-2500K (overclocked to 5GHz)

BOBBY R. BRUCE
INSPIRATION

Samsung Galaxy S7

Mali-T880 MP12

Intel i7-2500K (overclocked to 5GHz)

70 GFLOPs

BOBBY R. BRUCE
INSPIRATION

Samsung Galaxy S7

Mali-T880 MP12
265.2 GFLOPs

Intel i7-2500K (overclocked to 5GHz)
70 GFLOPs

BOBBY R. BRUCE
<table>
<thead>
<tr>
<th><strong>INSPIRATION</strong></th>
<th><strong>nVidia GTX 1060</strong></th>
<th><strong>Mali-T880 MP12</strong></th>
<th><strong>Intel i7-2500K (overclocked to 5GHz)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4327 GFLOPs</td>
<td>265.2 GFLOPs</td>
<td>70 GFLOPs</td>
</tr>
</tbody>
</table>

**BOBBY R. BRUCE**
WHY DON’T WE UTILISE THIS POWERFUL HARDWARE?

• Developers lack the skills

• Hardware specialisation

• Developers’ time is expensive; translating code to run on the GPU is expensive

• Getting decent optimisation requires manual trial and error

BOBBY R. BRUCE
WHY DON’T WE UTILISE THIS POWERFUL HARDWARE?

• Developers lack the skills
• Hardware specialisation
• Developers’ time is expensive; translating code to run on the GPU is expensive
• Getting decent optimisation requires manual trial and error

An Automated approach would be ideal

BOBBY R. BRUCE
BACKGROUND: WHAT'S CURRENTLY AVAILABLE?
<table>
<thead>
<tr>
<th>Domain</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>

BOBBY R. BRUCE
<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
</tr>
<tr>
<td>Cons</td>
<td>Only targets very specific loops where dependencies are fully understood</td>
</tr>
</tbody>
</table>
### BACKGROUND: WHAT’S CURRENTLY AVAILABLE?

<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
<th>CUDA/OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
<td>When implemented well offers the best performance</td>
</tr>
<tr>
<td>Cons</td>
<td>Only targets very specific loops where dependencies are fully understood</td>
<td>Difficult to learn, harder to master. Very Manual</td>
</tr>
</tbody>
</table>
## BACKGROUND: WHAT’S CURRENTLY AVAILABLE?

<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
<th>CUDA/OpenCL</th>
<th>Directive-based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
<td>When implemented well offers the best performance</td>
<td>Considerably easier to implement.</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>Only targets very specific loops where dependencies are fully understood</td>
<td>Difficult to learn, harder to master.</td>
<td>Still requires some skill, practise, and trial and error.</td>
</tr>
</tbody>
</table>

**BOBBY R. BRUCE**
## BACKGROUND: WHAT’S CURRENTLY AVAILABLE?

<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
<th>CUDA/OpenCL</th>
<th>Directive-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
<td>When implemented well offers the best performance</td>
<td>Considerably easier to implement.</td>
</tr>
<tr>
<td>Cons</td>
<td>Only targets very specific loops where dependencies are fully understood</td>
<td>Difficult to learn, harder to master.</td>
<td>Still requires some skill, practise, and trial and error.</td>
</tr>
</tbody>
</table>
**BACKGROUND: WHAT’S CURRENTLY AVAILABLE?**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
<th>CUDA/OpenCL</th>
<th>Directive-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
<td>When implemented well offers the best performance</td>
<td>Considerably easier to implement.</td>
</tr>
<tr>
<td>Cons</td>
<td>Only targets very specific loops where dependencies are fully understood</td>
<td>Difficult to learn, harder to master.</td>
<td>Still requires some skill, practise, and trial and error.</td>
</tr>
</tbody>
</table>
**BACKGROUND: WHAT’S CURRENTLY AVAILABLE?**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Automatic Parallelisation Compilers</th>
<th>CUDA/OpenCL</th>
<th>Directive-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Does not require any skills, or knowledge of, parallelisation</td>
<td>When implemented well offers the best performance</td>
<td>Considerably easier to implement.</td>
</tr>
<tr>
<td>Cons</td>
<td>Only targets very specific loops where dependencies are fully understood</td>
<td>Difficult to learn, harder to master.</td>
<td>Still requires some skill, practise, and trial and error.</td>
</tr>
</tbody>
</table>

**BOBBY R. BRUCE**
```c
while ( error > tol && iter < iter_max ) {
    error = 0.0;
    for( int j = 1; j < n-1; j++ ) {
        for(int i=1; i<m-1; i++) {
            A[j][i] = 0.25 * ( Anew[j][i+1] + Anew[j][i-1] 
                             + Anew[j-1][i] + Anew[j+1][i] );
            error = fmax( error, fabs(A[j][i] - Anew[j][i]) );
        }
    }
    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
    if(iter % 100 == 0){
        printf("%5d, %0.6f
", iter, error); iter++;
    }
    iter++;
}
```
#pragma acc data copy(A[1:n][1:m]) create(Anew[n][m])
while (error > tol && iter < iter_max) {
    error = 0.0;
    #pragma acc parallel loop reduction(max:error)
    for (int j = 1; j < n-1; j++) {
        #pragma acc loop reduction(max:error)
        for (int i=1; i<m-1; i++) {
            error = fmax(error, fabs(A[j][i] - Anew[j][i]));
        }
    }
    #pragma acc parallel loop
    for (int j = 1; j < n-1; j++) {
        #pragma acc loop
        for (int i = 1; i < m-1; i++) {
            A[j][i] = Anew[j][i];
        }
    }
    if (iter % 100 == 0) {
        printf("%5d, %0.6f\n", iter, error); iter++;
    }
}
iter++;
BACKGROUND: OPENACC

```c
#pragma acc data copy(A[1:n][1:m]) create(Anew[n][m])
while ( error > tol && iter < iter_max ) {
    error = 0.0;
#pragma acc parallel loop reduction(max:error)
    for( int j = 1; j < n-1; j++) {
        #pragma acc loop reduction(max:error)
        for(inti=1;i<m-1;i++) {
            A[j][i] = 0.25 * ( Anew[j][i+1] + Anew[j][i-1]
                              + Anew[j-1][i] + Anew[j+1][i]);
            error = fmax( error, fabs(A[j][i] - Anew[j][i]));
        }
    }
#pragma acc parallel loop
    for( int j = 1; j < n-1; j++) {
        #pragma acc loop
        for( int i = 1; i < m-1; i++) {
            A[j][i] = Anew[j][i];
        }
    }
    if(iter % 100 == 0){
        printf("%5d, %0.6f\n", iter, error); iter++;
    }
}
iter++;
```
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

Creates

Patch

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

CFG-GP

Creates

Patch

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

CFG-GP

Fitness Function

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

GRAMMAR

CFG-GP

creates

FITNESS FUNCTION
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

OPENACC GRAMMAR

GRAMMAR

CFG-GP

Fitness Function

Patch

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

OPENACC_GI

OPENACC GRAMMAR
PROGRAM DATA

GRAMMAR

CFG-GP

FITNESS FUNCTION

Creates

Patch

Patch

BOBBY R. BRUCE
OUR GOAL: AUTOMATICALLY ADD OPENACC DIRECTIVES

- OPENACC_GI
  - OPENACC GRAMMAR
  - PROGRAM DATA
  - GRAMMAR
  - CFG-GP

- LEXICAL ANALYSER
- SOURCE CODE

- FITNESS FUNCTION

- Patch

BOBBY R. BRUCE
GRAMMAR

\[
\text{<start>} ::= \text{<base>} | \text{<base>} \text{<start>}
\]
\[
\text{<base>} ::= "\#pragma acc " \text{<choice>}
\]
\[
\text{<choice>} ::= "\text{loop} " \text{<private>} \text{<loop line number>}
\]
\[
\text{<private>} ::= "\text{private(} " \text{<variables>} "\)} " | " "
\]
\[
\text{<variables>} ::= \text{<variable>} | \text{<variable>} , " \text{<variables>}
\]
\[
\text{<variable>} ::= \text{<variable placeholder>}
\]
\[
\text{<variable placeholder>} ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" ... 
\]
<start> ::= <base> | <base> <start>
<base> ::= "#pragma acc " <choice>
<choice> ::= "loop " <private> <loop_line_number>
<private> ::= "private(" <variables> ")" | ""
<variables> ::= <variable> | <variable> "," <variables>
<variable> ::= <variable_placeholder>
(variable_placeholder) ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" ...
(loop_line_number) ::= "15@example1.c" | "145@example2.c"
<start> ::= <base> | <base> <start>
<base> ::= "#pragma acc " <choice>
<choice> ::= "loop " <private> <loop_line_number>
<private> ::= "private(" <variables> ") " | " "
:variables> ::= <variable> | <variable> "," <variables>
<variable> ::= <variable_placeholder>
<variable_placeholder> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" …
<loop_line_number> ::= "15@example1.c" | "145@example2.c"

#pragma acc loop private(1,2) 15@example1.c
<start> ::= <base> | <base> <start>
;base> ::= "#pragma acc " <choice>
<choice> ::= "loop " <private> <loop_line_number>
<private> ::= "private(" <variables> ")" " | " "
<variables> ::= <variable> | <variable> "," <variables>
<variable> ::= <variable_placeholder>
<variable_placeholder> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" …
<loop_line_number> ::= "15@example1.c" | "145@example2.c"
GRAMMAR

<start> ::= <base> | <base> <start>
<base> ::= "#pragma acc " <choice>
(choice) ::= "loop " <private> <loop_line_number>
(private) ::= "private(" <variables> ")" " | " "
(variables) ::= <variable> | <variable> "," <variables>
(variable) ::= <variable_placeholder>
(variable_placeholder) ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" ...
(loop_line_number) ::= "_15@example1.c" | "145@example2.c"

--- example1.c
+++ example1.c
@@ -15,0 +15,1 @@
+ #pragma acc loop private(x,y)
INITIAL INVESTIGATION

• Chose to run a very small example as a sanity check

• nVidia provide an n-body simulation example already containing OpenACC directives

• These directives were stripped for openacc to replicate

• Ran for 100 generations with population of 100
RESULTS

Execution Time (ms)

- sequential
- original
- gi_best
RESULTS

Execution Time (ms)

- original
- gi_best

BOBBY R. BRUCE
RESULTS: OTHER NOTES

- Seems like much of the gain is due to random search
- We’d like to be able to beat human-written alternatives
- This example is very small, future investigations will show how well the tool scales
CURRENT/FUTURE WORK

• Currently applying the tool to larger

• At present can only work with C/C++, expanding code to work with FORTRAN

Possible Improvements:

• Seed initial generation with basic solutions

• Introduce some clever profiling

• Get working with OpenMP as well as OpenACC
ANY QUESTIONS?

EXECUTION TIME (ms)

- Sequential
- Original
- Optimal

ELITE PERFORMANCE (ms)

- Generation

BOBBY R. BRUCE