Exploring the fundamental differences between compiler optimisations for energy and for time

James Pallister and Kerstin Eder
University of Bristol
Compilers

Frontend -> Optimiser -> Backend
Compilers

Source program

Frontend → Optimiser → Backend
Compilers

Source program

Frontend → Optimiser → Backend

Compiled executable
Compilers

Large impact on program:
- runtime
- energy
- code size
Topics of interest
Topics of interest

How do compiler optimisations affect energy, power and time?
Topics of interest

How do compiler optimisations affect energy, power and time?

Are there different “types” of optimisation?
Topics of interest

How do compiler optimisations affect energy, power and time?
Are there different “types” of optimisation?
How does the composition of optimisations affect these metrics?
Topics of interest

How do compiler optimisations affect energy, power and time?
Are there different “types” of optimisation?
How does the composition of optimisations affect these metrics?
Compilers → automatic
Topics of interest

How do compiler optimisations affect energy, power and time?
Are there different “types” of optimisation?
How does the composition of optimisations affect these metrics?
Compilers → automatic
Embedded systems
Overview
Overview

Introduction
Overview

Introduction

Optimisations for execution time
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Conclusion
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Conclusion
An optimisation's effect

\[ E = P \times T \]
An optimisation's effect

\[ E = P \times T \]
An optimisation’s effect

\[ E = P \times T \]

Average power

Execution time
An optimisation's effect

\[
E = P \times T
\]

- Energy
- Execution time
- Average power
An optimisation's effect

\[ E = P \times T \]
An optimisation's effect

\[ E = P \times T \]

\[ T' = k_T \cdot T \]
An optimisation's effect

\[ E = P \times T \]

\[ T' = k_T \cdot T \]
An optimisation's effect

\[ E = P \times T \]

Execution time after applying the optimisation

\[ T' = k_T \cdot T \]

Optimisation's effect on time
An optimisation's effect

\[ E = P \times T \]

\[ T' = k_T \cdot T \]
An optimisation's effect

\[ E = P \times T \]

\[ T' = k_T \cdot T \]

\[ P' = k_P \cdot P \]
An optimisation's effect

\[ E = P \times T \]

\[ T' = k_T \cdot T \]

\[ P' = k_P \cdot P \]

\[ E' = (k_P \cdot P) \times (k_T \cdot T) \]
An optimisation's effect

\[ T' = k_T \cdot T \]

\[ P' = k_P \cdot P \]

\[ E' = (k_P \cdot P) \times (k_T \cdot T) \]
An optimisation's effect

Energy is saved if:

\[(k_T \cdot k_P) < 1\]

\[T' = k_T \cdot T\]

\[P' = k_P \cdot P\]

\[E' = (k_P \cdot P) \times (k_T \cdot T)\]
An optimisation's effect

\[ T' = k_T \cdot T \]

\[ P' = k_P \cdot P \]

\[ E' = (k_P \cdot P) \times (k_T \cdot T) \]
An optimisation's effect

Hypothesis:
The existing optimisations in the compiler only affect energy by reducing $k_T$.

\[
T' = k_T \cdot T
\]

\[
P' = k_P \cdot P
\]

\[
E' = (k_P \cdot P) \times (k_T \cdot T)
\]
An optimisation's effect
An optimisation's effect

Hypothesis:
The existing optimisations in the compiler only affect energy by reducing $k_T$. 
An optimisation's effect

Hypothesis:
The existing optimisations in the compiler only affect energy by reducing $k_T$.

Optimisations in the compiler have been designed for reducing execution time.
Optimisation example

![Diagram showing optimisation example with categories Multiply, Stall, and Add.]
Optimisation example

\[ k_T = 0.8 \]
\[ k_P = 1.14 \]
\[ k_E = k_T \cdot k_P = 0.91 \]
Optimisation example

$$k_T = 0.8$$
$$k_P = 1.14$$
$$k_E = k_T \cdot k_P = 0.91$$

$$k_T = 1.0$$
$$k_P = 0.82$$
$$k_E = k_T \cdot k_P = 0.82$$
Interactions and ordering
Interactions and ordering

No transformations

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
        * function1(b + 1);
}
```
Interactions and ordering

No transformations

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
        * function1(b + 1);
}
```

CSE

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```
Interactions and ordering

No transformations

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1) * function1(b + 1);
}
```

CSE

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```c
int function2(int b)
{
    return ((b + 1) * (b + 1)) * ((b + 1) * (b + 1));
}
```
Interactions and ordering

No transformations

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
    * function1(b + 1);
}
```

CSE

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```c
int function2(int b)
{
    return ((b + 1) * (b + 1))
    * ((b + 1) * (b + 1));
}
```

CSE, then inlining

```c
int function2(int b)
{
    int t1 = (b + 1) * (b + 1);
    return t1 * t1;
}
```
Interactions and ordering

No transformations

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    return function1(b + 1)
    * function1(b + 1);
}
```

CSE

```c
int function1(int a)
{
    return a * a;
}

int function2(int b)
{
    int t1 = function1(b + 1);
    return t1 * t1;
}
```

Inlining

```c
int function2(int b)
{
    return ((b + 1) * (b + 1))
    * ((b + 1) * (b + 1));
}
```

CSE, then inlining

```c
int function2(int b)
{
    int t1 = (b + 1) * (b + 1);
    return t1 * t1;
}
```

Inlining, then CSE

```c
int function2(int b)
{
    int t1 = b + 1;
    return (t1 * t1) * (t1 * t1);
}
Research questions
Research questions

Standard compiler optimisations

- Do existing compiler optimisations save energy purely by reducing the coefficient $k_T$?
- Are there any optimisations which affect $k_P$?
Research questions

**Standard compiler optimisations**
- Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?
- Are there any optimisations which affect $k_P$?

**Compiler optimisations for energy**
- Is there a class of optimisations which can lower the energy via the $k_P$ coefficient?
- How are they different?
- Are they effective?
Research questions

Combining optimisations

- Do optimisations designed to lower $k_P$ interact with existing compiler optimisations?
- Are different sets of optimisations effective when including energy optimisations?
Overview

Introduction

- Optimisations for execution time
  Optimisations for energy consumption
  Combining optimisations for energy and time

Conclusion
Optimisations for time
Optimisations for time

Optimisations already in the compiler
Optimisations for time

Optimisations already in the compiler
Typically reducing code
- Common subexpression elimination
- Dead code elimination
Optimisations for time

Optimisations already in the compiler

Typically reducing code
- Common subexpression elimination
- Dead code elimination

Typically reordering code
- Instruction scheduling
- Array access patterns
- Moving invariant code out of loops
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy.

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

- Individual optimisations

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

- Individual optimisations
- Sets of optimisations

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Optimisations for time

Test how the compiler optimisations affect energy, by measuring both time and energy

- Individual optimisations
- Sets of optimisations
- The “best” set of optimisations

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Individual optimisations

“How existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
"Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?"
“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Sets of optimisations

No single set of optimisations is good for all benchmarks or all SoCs.

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
No single set of optimisations is good for all benchmarks or all SoCs.
The set of effective optimisations for each benchmark is different

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Sets of optimisations

No single set of optimisations is good for all benchmarks or all SoCs.

The set of effective optimisations for each benchmark is different

- But the same for time and energy

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
The best set

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

“Are there any optimisations which affect $k_P$?”
The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

“Are there any optimisations which affect $k_P$?”
The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

- Genetic algorithms

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

“Are there any optimisations which affect $k_P$?”
The best set

Attempt to pick good combinations of optimisations which maximise a fitness function
- Genetic algorithms

Compare the results of finding the best set for energy and for time.

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
“Are there any optimisations which affect $k_P$?”
The best set

Attempt to pick good combinations of optimisations which maximise a fitness function

- Genetic algorithms

Compare the results of finding the best set for energy and for time.

- If similar, most of the energy savings are coming from a reduction in time.

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

“Are there any optimisations which affect $k_P$?”
The best set

Minimise energy. \( F(e, t) = \frac{1}{e} \).

Minimise time. \( F(e, t) = \frac{1}{t} \).

Minimise power. \( F(e, t) = \frac{e}{t} \).

Maximise power. \( F(e, t) = \frac{t}{e} \).

“Do existing compiler optimisations save energy purely by reducing the \( k_T \) coefficient?”

“Are there any optimisations which affect \( k_P \)?”
“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

“Are there any optimisations which affect $k_P$?”
Optimisations for time
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

- Yes, but some cases where power gets lowered also
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

- Yes, but some cases where power gets lowered also
- Not significantly
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”

– Yes, but some cases where power gets lowered also
– Not significantly

“Are there any optimisations which affect $k_P$ ?”
Optimisations for time

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
- Yes, but some cases where power gets lowered also
- Not significantly

“Are there any optimisations which affect $k_P$?”
- Not significantly
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Conclusion
Optimisations for energy
Optimisations for energy

Try to find some optimisations which change the power coefficient
Optimisations for energy

Try to find some optimisations which change the power coefficient

Exploit some characteristics of the target processors:

- Embedded flash structure
- Flash and RAM differences
Embedded flash energy
Embedded flash energy

Flash on the same die as the rest of the SoC
Embedded flash energy

Flash on the same die as the rest of the SoC
Embedded flash energy

Flash on the same die as the rest of the SoC

Code execution directly from flash
Embedded flash energy

Flash on the same die as the rest of the SoC

Code execution directly from flash

Structure of embedded flash is non-uniform
Embedded flash memory

[Diagram of flash memory architecture with labeled components: Word-lines, Bit-lines, Block, Decoder, Address bus, Bus to CPU]
Expected energy effect

Decoder

Block$_{m-1}$

$B_0$  $B_{n-1}$

Word-lines

Bit-lines

$S_0$

$W_0$

$W_{k-4}$

$W_{k-3}$

$W_{k-2}$

$W_{k-1}$

$S_1$

Block$_0$

$S_0$

$W_0$

$W_1$

$W_{k-1}$

$S_1$
Expected energy effect

```
loop:
nop
nop
subs r0, #1
bne loop
```
Expected energy effect

n-bit length instructions

loop: nop

nop

subs r0, #1

bne loop
Expected energy effect

Decoders

Word-lines

Bit-lines

Block_{m-1}

Block_0

n-bit length instructions

Straddling two blocks.

loop: nop
nop
subs r0, #1

bne loop
Expected energy effect

n-bit length instructions

Straddling two blocks.
Each iteration must repeatedly power up one, then the other
Expected energy effect

Straddling two blocks.

Each iteration must repeatedly power up one, then the other

Higher energy cost when an instruction jumps from one 'region' to another.
Expected energy effect

Straddling two blocks.
Each iteration must repeatedly power up one, then the other.
Higher energy cost when an instruction jumps from one 'region' to another.

n-bit length instructions

loop: nop
nop

subs r0, #1
bne loop
Actual results

Bottom line (blue) → 8-byte loop

Top line (green) → 10-byte loop
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. \(0 \rightarrow 2\)
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2

[Diagram showing consecutive memory accesses and energy consumption]
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

1-byte
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2

\[ E_0 + E_1 \]
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

$E_0 + E_1$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. \(0 \rightarrow 2\) \(E_0 + E_1\)
Modelling

$E_0 + E_1$
Modelling

$E_0 + E_1$
Modelling

\[ E_0 + E_1 \]
Modelling

\[ E_0 + E_1 \]

1-byte

\[ E_0 \]
Modelling

\[ E_0 + E_1 \]

\[ E_0 \]

2-byte
Modelling

\[ E_0 + E_1 \]

2-byte

\[ E_0 + E_1 \]
Modelling

$E_0 + E_1$

4-byte

$E_0 + E_1$
Modelling

\[ E_0 + E_1 \]

\[ E_0 + E_1 + E_2 \]
Modelling

\[ E_0 + E_1 \]

8-byte

\[ E_0 + E_1 + E_2 \]
Cross validation

STM32F0
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
A potential optimisation

Unfortunately it doesn't save energy

Overhead of aligning loops outweighs the potential benefit

- Many boundary crossings are necessary
- Smaller boundary crossings happen more frequently

More sophisticated implementation may be able to save energy
Flash or RAM
Flash or RAM

Execute instructions from

- Flash
- RAM
Flash or RAM

Execute instructions from

Measure the power dissipation
Flash or RAM

Execute instructions from

Measure the power dissipation

![Bar chart showing power dissipation for different types of instructions compared between Flash and RAM.]
Flash or RAM

![Bar chart showing power consumption of flash and RAM for different instruction types.](chart.png)
Flash or RAM

; r1 in RAM
str r0, [r1]
Flash or RAM

; r1 in RAM
str r0, [r1]

; r1 in RAM
ldr r0, [r1]
Flash or RAM

; r1 in RAM
str r0, [r1]

add r0, r1

; r1 in RAM
ldr r0, [r1]

<table>
<thead>
<tr>
<th>Type of instruction</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ram store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ram load</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flash load</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Flash or RAM

; r1 in RAM
str  r0, [r1]

ldr  r0, [r1]
add  r0, r1

; r1 in RAM

nop

---

![Bar chart comparing power consumption between Flash and RAM](chart.png)

<table>
<thead>
<tr>
<th>Type of instruction</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ram store</td>
<td>Flash: 12</td>
</tr>
<tr>
<td></td>
<td>RAM: 6</td>
</tr>
<tr>
<td>ram load</td>
<td>Flash: 14</td>
</tr>
<tr>
<td></td>
<td>RAM: 8</td>
</tr>
<tr>
<td>add</td>
<td>Flash: 10</td>
</tr>
<tr>
<td></td>
<td>RAM: 5</td>
</tr>
<tr>
<td>nop</td>
<td>Flash: 16</td>
</tr>
<tr>
<td></td>
<td>RAM: 9</td>
</tr>
<tr>
<td>branch</td>
<td>Flash: 18</td>
</tr>
<tr>
<td></td>
<td>RAM: 11</td>
</tr>
<tr>
<td>flash load</td>
<td>Flash: 14</td>
</tr>
<tr>
<td></td>
<td>RAM: 7</td>
</tr>
</tbody>
</table>
Flash or RAM

; r1 in RAM
str r0, [r1]

; r1 in RAM
ldr r0, [r1]

add r0, r1

nop

b label

<table>
<thead>
<tr>
<th>Type of instruction</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ram store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ram load</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flash load</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Flash or RAM

; r1 in RAM
str r0, [r1]

str r0, [r1]

; r1 in flash
ld r0, [r1]

ldr r0, [r1]

add r0, r1

b label

nop

ld r0, [r1]
Move basic blocks into RAM

Original code

```c
int fn(int k)
{
    int i, x;
    x = 1;
    for(i = 0; i < 64; ++i)
    {
        x *= k;
    }
    if(x > 255)
    {
        x = 255;
    }
    return x;
}
```
Move basic blocks into RAM

Original code

```c
int fn(int k)
{
    int i, x;
    x = 1;
    for(i = 0; i < 64; ++i)
    {
        x *= k;
    }
    if(x > 255)
    {
        x = 255;
    }
    return x;
}
```

Original compiled code

```
init
    mov r0, #0
    mov r1, #1

loop
    mul r1, r1, r2
    add r0, r0, #1
    cmp r0, #64
    bne loop

if
    cmp r1, #255
    bge return

iftrue
    return

return
    mov r0, #255
    bx lr
```
Move basic blocks into RAM

Original code

```c
int fn(int k)
{
    int i, x;
    x = 1;
    for(i = 0; i < 64; ++i)
    {
        x += k;
    }
    if(x > 255)
        x = 255;
    return x;
}
```

Original compiled code

```
init
mov r0, #0
mov r1, #1
loop
mul r1, r1, r2
add r0, r0, #1
cmp r0, #64
bne loop
if
cmp r1, #255
ble return
iftrue
return
mov r0, #255
return
mov r0, r1
bx lr
```

Optimized compiled code

```
init
mov r0, #0
mov r1, #1
ldr pc, =loop
loop
mul r1, r1, r2
add r0, r0, #1
cmp r0, #64
bne loop
if
cmp r1, #255
it le
ldrle r5, =return
ldrgt r5, =iftrue
bx r5
iftrue
mov r0, #255
return
mov r0, r1
bx lr
```

Additional long range branches are needed to jump between memories.

Instrumenting the if block instead of the loop block reduces the overall energy/time.
Optimisations for energy

The RAM overlay can save significant energy
- Up to 26%

This is achieved by reducing the $k_P$ coefficient
- The execution time actually increases.

Aligning to flash boundaries is currently ineffective.
- Many of the boundary changes are necessary
- Other types of optimisations may succeed
Overview

Introduction

Optimisations for execution time

Optimisations for energy consumption

Combining optimisations for energy and time

Conclusion
Combining optimisations
Combining optimisations

![Graph showing execution time vs. average power with annotations for 'Worst: 6.7 mJ' and 'Best: 2.6 mJ']
Combining optimisations
Combining optimisations

![Graph showing execution time vs average power with various energy consumption scenarios. The graph indicates that the lowest energy found is 2.6 mJ (best case) and the highest energy found is 6.7 mJ (worst case).]
Combining optimisations

![Graph showing performance metrics with annotations for worst and best cases.]

- **Worst:** 6.7 mJ
- **Best:** 2.6 mJ

Legend:
- Lowest energy found
- Highest energy found
- Energy for 0.3
- Energy for 0.3 + RAM overlay
- RAM Overlay disabled
- RAM Overlay enabled
Fractional factorial design again
Fractional factorial design again
Composability

![Composability Diagram]

- Energy
- Time

Benchmark:
- 2dfir
- blowfish
- crc32
- cubic
- dijkstra
- fdct
- matmult-float
- matmult-int
- rijndael
- sha
Composability
Composability
Composability
Composability
Combining optimisations

- The optimisations do not significantly interact.

- The same set of “base” optimisations can be applied
  • i.e. we do not need a new optimisation level to apply energy on top of.
Conclusion
Conclusion

Existing optimisations affect execution time by design.
Conclusion

Existing optimisations affect execution time by design.

Existing optimisations affect power by chance.
Conclusion

Existing optimisations affect execution time by design.

Existing optimisations affect power by chance.

Optimisations for energy do exist, they reduce energy by reducing average power.
Conclusion

Existing optimisations affect execution time by design.
Existing optimisations affect power by chance.

Optimisations for energy do exist, they reduce energy by reducing average power.

These optimisations combine linearly with existing optimisations.
Future work
Future work

How to find new energy optimisations?
Future work

How to find new energy optimisations?
Do all energy optimisations combine linearly?
Future work

How to find new energy optimisations?
Do all energy optimisations combine linearly?
Do the findings apply outside of embedded systems?
Future work

How to find new energy optimisations?
Do all energy optimisations combine linearly?
Do the findings apply outside of embedded systems?
Does the efficacy of the optimisations change when adding more cores/threads?
Thanks!

Questions?

More info:


Embedded systems
# Embedded systems

<table>
<thead>
<tr>
<th>Board Name</th>
<th>SoC Processor</th>
<th>RAM</th>
<th>ROM</th>
<th>Clock</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F0DISCOVERY</td>
<td>STM32F051</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>Cortex-M0</td>
<td>8kB</td>
<td>64kB</td>
<td></td>
<td>LLVM ✓</td>
</tr>
<tr>
<td>STM32VLDISCOVERY</td>
<td>STM32F100</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>Cortex-M3</td>
<td>8kB</td>
<td>128kB</td>
<td></td>
<td>LLVM ✓</td>
</tr>
<tr>
<td>Breadboard</td>
<td>ATMEGA328P</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>AVR8</td>
<td>2kB</td>
<td>32kB</td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>XMEGA-A3BU Xplained</td>
<td>ATXMEGA256A3BU</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>AVR8</td>
<td>16kB</td>
<td>256kB</td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>Breadboard</td>
<td>PIC32MX250F128B</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>MIPS32</td>
<td>32kB</td>
<td>128kB</td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>MSP-EXP430F5229LP</td>
<td>MSP430F5529</td>
<td>SRAM</td>
<td>Flash</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>MSP430</td>
<td>10kB</td>
<td>128kB</td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>MSP-EXP430FR5739LP</td>
<td>MSP430FR5739</td>
<td>SRAM</td>
<td>FRAM</td>
<td>8MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>MSP430</td>
<td>1kB</td>
<td>16kB</td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>Beagle Bone</td>
<td>AM335x</td>
<td>DRAM</td>
<td>SD-Card</td>
<td>800MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>Cortex-A8</td>
<td>256MB</td>
<td></td>
<td></td>
<td>LLVM ✓</td>
</tr>
<tr>
<td>EMEK4 Dev Board</td>
<td>EMEK4</td>
<td>SRAM</td>
<td>None</td>
<td>800MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>Epiphany</td>
<td>32kB</td>
<td></td>
<td></td>
<td>LLVM ×</td>
</tr>
<tr>
<td>XK-1A</td>
<td>XS1-L8-64</td>
<td>SRAM</td>
<td>None</td>
<td>400MHz</td>
<td>GCC ✓</td>
</tr>
<tr>
<td></td>
<td>XMOS-XS1</td>
<td>64k</td>
<td></td>
<td></td>
<td>LLVM ✓</td>
</tr>
</tbody>
</table>
Individual optimisations

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Individual optimisations

Fractional factorial design

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Individual optimisations

Fractional factorial design

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
Individual optimisations

Fractional factorial design

“Do existing compiler optimisations save energy purely by reducing the $k_T$ coefficient?”
## Parameters

<table>
<thead>
<tr>
<th>SoC</th>
<th>$E_2$ (A)</th>
<th>$E_3$</th>
<th>$E_4$ (B)</th>
<th>$E_5$</th>
<th>$E_6$</th>
<th>$E_7$ (C)</th>
<th>$E_8$ (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F0</td>
<td>300</td>
<td>27</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>STM32F1</td>
<td>500</td>
<td>0</td>
<td>6</td>
<td>34</td>
<td>4</td>
<td>10</td>
<td>190</td>
</tr>
<tr>
<td>ATMEGA328P</td>
<td>0</td>
<td>22</td>
<td>36</td>
<td>27</td>
<td>9</td>
<td>107</td>
<td>24</td>
</tr>
<tr>
<td>PIC32MX250F128B</td>
<td>225</td>
<td>0</td>
<td>10</td>
<td>18</td>
<td>8</td>
<td>13</td>
<td>113</td>
</tr>
<tr>
<td>MSP430F5529</td>
<td>408</td>
<td>0</td>
<td>34</td>
<td>26</td>
<td>15</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>SoC</th>
<th>$E_2$ (A)</th>
<th>$E_3$</th>
<th>$E_4$ (B)</th>
<th>$E_5$</th>
<th>$E_6$</th>
<th>$E_7$ (C)</th>
<th>$E_8$ (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F0</td>
<td>300</td>
<td>27</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>STM32F1</td>
<td>500</td>
<td>0</td>
<td>6</td>
<td>34</td>
<td>4</td>
<td>10</td>
<td>190</td>
</tr>
<tr>
<td>ATMEGA328P</td>
<td>0</td>
<td>22</td>
<td>36</td>
<td>27</td>
<td>9</td>
<td>107</td>
<td>24</td>
</tr>
<tr>
<td>PIC32MX250F128B</td>
<td>225</td>
<td>0</td>
<td>10</td>
<td>18</td>
<td>8</td>
<td>13</td>
<td>113</td>
</tr>
<tr>
<td>MSP430F5529</td>
<td>408</td>
<td>0</td>
<td>34</td>
<td>26</td>
<td>15</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>SoC</th>
<th>$E_2$ (A)</th>
<th>$E_3$</th>
<th>$E_4$ (B)</th>
<th>$E_5$</th>
<th>$E_6$</th>
<th>$E_7$ (C)</th>
<th>$E_8$ (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F0</td>
<td>300</td>
<td>27</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>STM32F1</td>
<td>500</td>
<td>0</td>
<td>6</td>
<td>34</td>
<td>4</td>
<td>10</td>
<td>190</td>
</tr>
<tr>
<td>ATMEGA328P</td>
<td>0</td>
<td>22</td>
<td>36</td>
<td>27</td>
<td>9</td>
<td>107</td>
<td>24</td>
</tr>
<tr>
<td>PIC32MX250F128B</td>
<td>225</td>
<td>0</td>
<td>10</td>
<td>18</td>
<td>8</td>
<td>13</td>
<td>113</td>
</tr>
<tr>
<td>MSP430F5529</td>
<td>408</td>
<td>0</td>
<td>34</td>
<td>26</td>
<td>15</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

![Graph](image.png)