Optimized Unrolling of Nested Loops

Ian Iong Lam, Jiayao Su, Tony Tang

01

MOTIVATION

area + problem + why is it important to solve this problem

Loop Unrolling Benefits

OVERHEAD

Reduce amortized increment-and-test overhead

PARALLELISM

increased instruction-level parallelism

REGISTER

improved register locality

MEMORY

improved I-cache performance

Optimized Unrolling of Nested Loops

Nested Loops

Able to unroll nested loops

Unroll Vectors

Efficiently find feasible unroll vectors

Cost Model

includes ILP and I-cache consideration

Compact Loops

New code generation algo to generate more compact loops

02

TECHNIQUE

Nested loops unrolling algorithm explanation

Unroll Nested Loops Technique

Unroll Vector Selection

1. Unroll vectors search
2. New cost model calculation

Code Generation

1. Unroll perfectly nested loops from outermost to innermost
2. Allow high-order optimizations such as LICM

Unroll Vector

There are **k perfectly nested loops** where loop 1 is the outermost loop

UF_i is an integer in range **$[1, UF_i^{max}]$**
where **UF_i^{max}** is the **number of iterations** for loop i

$$(UF_1, UF_2, UF_3, \dots UF_k)$$

Unroll factor for loop 1

Unroll Vector Search

Capacity Constraint

1. The amortized #register spills in the unrolled loop body does not exceed the original #register spills

2. Size of unrolled loop body fits in the I-cache

Find Feasible Vectors

Input: cur loop idx **i**, cur unroll vector $\mathbf{UV_{cur}}$
Output: feasible unroll vectors **UVs**

Find Algo find(i, $\mathbf{UV_{cur}}$):
For n = 1 to UF_i^{max}
1. Update current vector $\mathbf{UV_{cur}}$ with n at idx **i**
2. Break if $\mathbf{UV_{cur}}$ exceeds capacity constraint
3. if **i** = 1, add $\mathbf{UV_{cur}}$ to **UVs** (1 vector found)
4. else find(**i-1**, $\mathbf{UV_{cur}}$)

Cost Function for Unroll Vector Optimization

uv = (UF_1, ..., UF_k) = current unroll vector
LS(uv) = EST. # cycles spend on Load and Store instr in unrolled loop body
CP(uv) = EST. critical path length (in cycles) of unrolled loop body
TC_j(uv) = EST. total cycles on a class j of functional unit required by unrolled loop body
NF_j = # functional unit of class j avail in the hardware
F(uv) = Cost function

$$F(\mathrm{uv}) = \overbrace{\frac{LS(\mathrm{uv})}{UF_1 \times \cdots \times UF_k}}^{\text{load/store term}} + \overbrace{\frac{\max[CP(\mathrm{uv}), \max_j \left(\left\{\frac{TC_j(\mathrm{uv})}{NF_j}\right\}\right)]}{UF_1 \times \cdots \times UF_k}}^{\text{ILP term}}$$

Optimize Unroll Vector

Input: Feasible unroll vectors candidate pool **UVs**
Output: $\mathbf{UV}^{\mathbf{opt}} = (UF_1^{opt}, \ldots, UF_k^{opt})$, and optimized unroll vector for input nested loops

Optimize Algo opt(UVs):
Initialize $\mathbf{UV}^{\mathbf{opt}} = (1, \ldots, 1)$
For **uv** in **UV**

1. if $F(\mathbf{uv}) < F(\mathbf{UV}^{\mathbf{opt}})$ or $(F(\mathbf{uv}) < F(\mathbf{UV}^{\mathbf{opt}})$ and $(\mathbf{uv}_1 \times \ldots \times \mathbf{uv}_k) < (\mathbf{UV}_k^{\mathbf{opt}} \times \ldots \times \mathbf{UV}_k^{\mathbf{opt}}))$
2. then $\mathbf{UV}^{\mathbf{opt}} = \mathbf{uv}$

Code Generation

Code Generation Algorithm:

For each nested loop **i**:

1. Make copies of the inner loop body according to the unroll factor $\mathbf{UF_i}$
2. Adjust header information such as lower bound, upper bound, and increments
3. Construct remainder loops

Code Generation Example

Outer Loop

```
for (i=0; i < 10; i++) {
    for(j=0; j < 10; j++) {
        Sum += a[i] * b[j];
    }
}
```

Inner Loop

Inner loop:

```
for(j=0; j < 10; j++) {
    Sum += a[i] * b[j];
}
```

unroll inner loop:

Inner Unrolled Body for idx i

```
for(j=0; j < 10; j+=4) {
    Sum += a[i] * b[j];
    Sum += a[i] * b[j+1];
    Sum += a[i] * b[j+2];
    Sum += a[i] * b[j+3];
}
```

unroll with unroll factor = 4

```
for(j=8; j < 10; j++) {
    Sum += a[i] * b[j];
}
```

remainder loop

Unroll Vector = (3, 4)

Code Generation Example

Original Nested Loop

```
for (i=0; i < 10; i++) {
    for(j=0; j < 10; j++) {
        Sum += a[i] * b[j];
    }
}
```

Unroll Inner Loop

```
for (i=0; i < 10; i++) {
    Inner Unrolled Body for idx i
}
```

Unroll Outer Loop

```
for (i=0; i < 10; i+=3) {
    Inner Unrolled Body for idx i
    Inner Unrolled Body for idx i + 1
    Inner Unrolled Body for idx i + 2
}

for (i=9; i < 10; i++) {
    Inner Unrolled Body for idx i
}
```

Unroll Vector = (3, 4)

Inner Unrolled Body for idx i

```
for(j=0; j < 10; j+=4) {
        Sum += a[i] * b[j];
        Sum += a[i] * b[j+1];
        Sum += a[i] * b[j+2];
        Sum += a[i] * b[j+3];
}

for(j=8; j < 10; j++) {
        Sum += a[i] * b[j];
}
```

03

RESULTS

Algorithm’s application and improvements in benchmarks

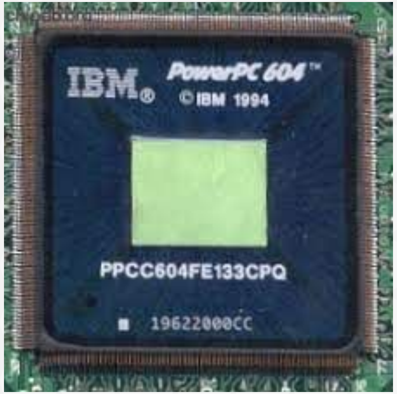
Benchmark & Hardware

SPEC95fp

- developed by the Standard Performance Evaluation Corporation (SPEC) to measure the floating-point performance of computer systems

IBM XL Fortran product compiler

133MHz PowerPC 604 processor



Results

Speedups (relative to NO-UNROLL) for different unroll configurations

Benchmark	NO-UNROLL	(2,2,2)	(3,3,3)	(4,4,4)	(5,5,5)	OPT-UNROLL
101.tomcatv	1.00	1.11	1.05	1.02	0.96	1.23
102.swim	1.00	1.04	0.86	0.75	0.73	1.20
103.su2cor	1.00	1.03	1.06	1.02	1.03	1.03
104.hydro2d	1.00	1.06	1.06	1.04	1.08	1.06
107.mgrid	1.00	1.05	0.99	0.96	0.72	1.00
125.turb3d	1.00	0.98	0.94	0.83	0.89	1.00
145.fpppp	1.00	0.99	0.97	1.01	0.80	1.02
Average Speedup	**1.00**	**1.04**	**0.99**	**0.95**	**0.89**	**1.08**

- OPT-UNROLL: algorithm reported in this paper
- Max: 1.2x; Average: 1.08x
- Never slower than NO-UNROLL

Conclusion

Pros

- **Able to unroll perfectly nested loops**
- **Automatically optimize unroll vector**
- **Neve slower than original program**
- Runtime reduction should work better on hardware with more registers and larger degrees of parallelism

Cons

- May not work for all nested loops
- Significantly increase code size

THANKS!

THANKS!

Do you have any questions?

iinicole@umich.edu
sujiayao@umich.edu
tonytang@umich.edu