Auto-Vectorization with GCC

Hanna Franzen  Kevin Neuenfeldt

HPAC
High Performance and Automatic Computing

Seminar on Code-Generation
1. Introduction
   - What Is Vectorization
   - What Are The Challenges

2. Structure Of GCC

3. Vectorization

4. Code Generation & Algorithm

5. Summary
Outline

1. Introduction
   - What Is Vectorization
   - What Are The Challenges

2. Structure Of GCC

3. Vectorization

4. Code Generation & Algorithm

5. Summary
What Is Vectorization

- **SIMD**: Single Instruction Multiple Data
- Most modern CPUs have SIMD vector registers of mostly 8 to 16 bytes
- Check support on UNIX (Intel/AMD):

  ```
  cat /proc/cpuinfo | grep -E \s(sse|mmx)\s
  ```

**Definition**

**Vectorization** is the rewriting of loops into vector instructions. Notation: $a[i:j] = \text{Elements between index } i \text{ and } j \text{ (including)}$. 
Challenges

- Memory references must (mostly) be consecutive
- Memory must be aligned to natural vector size boundary
- Number of iterations must be countable

Data Dependences
- Semantics of the vectorized program must be identical to the original program

Example

```c
for (int i=1; i<5; i++)
{
    a[i] = a[i-1] + b[i];  // S1
}
```

Assume a = [1,2,3,4,5] and b = [5,4,3,2,1].

- Sequential: a = [1,5,8,10,11]
- Vectorized: a = [1,5,5,5,5]
Challenges

- Memory references must (mostly) be consecutive
- Memory must be aligned to natural vector size boundary
- Number of iterations must be countable

Data Dependences

Semantics of the vectorized program must be identical to the original program

Example

```
for (int i=1; i<5; i++)
{
    a[i] = a[i-1] + b[i];  // S1
}
```

Assume `a = [1,2,3,4,5]` and `b = [5,4,3,2,1]`.

- Sequential: `a = [1,5,8,10,11]`
- Vectorized: `a = [1,5,5,5,5]`
Challenges

- Memory references must (mostly) be consecutive
- Memory must be aligned to natural vector size boundary
- Number of iterations must be countable
- Data Dependences
- Semantics of the vectorized program must be identical to the original program

Example

```c
for (int i=1; i<5; i++)
{
    a[i] = a[i-1] + b[i];  // S1
}
```

Assume `a = [1,2,3,4,5]` and `b = [5,4,3,2,1]`.

- Sequential: `a = [1,5,8,10,11]`
- Vectorized: `a = [1,5,5,5,5]`
Outline

1. Introduction
   - What Is Vectorization
   - What Are The Challenges

2. Structure Of GCC

3. Vectorization

4. Code Generation & Algorithm

5. Summary
Vector instructions are platform dependent
Different processors have different SIMD instruction sets
When built GCC uses machine description file to store which operations are available
Intermediate Languages (IL): RTL (Register Transfer Language)
IL: GIMPLE
Vector instructions are platform dependent
Different processors have different SIMD instruction sets
When built GCC uses machine description file to store which operations are available
Intermediate Languages (IL): RTL (Register Transfer Language)
IL: GIMPLE
Outline

1. Introduction
   - What Is Vectorization
   - What Are The Challenges

2. Structure Of GCC

3. Vectorization

4. Code Generation & Algorithm

5. Summary
Overview Vectorization Steps

Analyze...

- loop form: exit condition, control-flow, ...
- memory references: compute function that describes memory reference modifications
  - scalar dependence cycles
  - data reference access patterns
  - data reference alignment
  - data reference dependences
- Determine $VF$ (Vectorization Factor)
Overview Vectorization Steps

Analyze...

- loop form: exit condition, control-flow, ...
- memory references: compute function that describes memory reference modifications
- scalar dependence cycles
- data reference access patterns
- data reference alignment
- data reference dependences
- Determine \( VF \) (Vectorization Factor)
Scalar Dependences

- Use of scalar variables (no arrays, pointers)
- Example: Reduction

Example

```c
for (i=0; i<N; i++)
{
    sum += A[i];
}
```
Scalar Dependences

- Use of scalar variables (no arrays, pointers)
- Example: Reduction

Example

```c
for (i=0; i<N; i++)
{
    sum += A[i];
}
```
Scalar Dependences: Reduction

Example

```c
for (i=0; i<N; i++)
{
    sum += A[i];
}
```

- Intel Core 2Duo B950 → $VF = 4$
- $N = 1024$, 256 partial sums
Access Patterns

- Most SIMD instruction sets require consecutive memory access
- Otherwise permutations on data is needed
- Some SIMD instruction sets support certain access patterns (e.g. odd/even for complex numbers)
- Costs of permutations are critical to the worth of the vectorization

Example

```c
for (i=0; i<N; i++)
{
    A[i*2] = x;
}
```
Access Patterns

- Most SIMD instruction sets require consecutive memory access.
- Otherwise permutations on data is needed.
- Some SIMD instruction sets support certain access patterns (e.g. odd/even for complex numbers).
- Costs of permutations are critical to the worth of the vectorization.

Example

```c
for (i=0; i<N; i++)
{
    A[i*2] = x;
}
```
Alignment

Access on an unaligned memory location is needed.

Target platform needs at least one capability:

- memory read/write to unaligned memory location
- general vector shuffling ability
- specialized alignment support

Most SIMD platforms load VS bytes from closest previous aligned address (e.g. AltiVec, Alpha).
Data Dependences

\[ S_1 \] → \[ S_2 \] \( \delta^0 \) \( \delta_1 \) \( \delta^{-1}_1 \) \( \delta^{-1}_1 \) \( \delta_1 \) \( \delta_\infty \)

\[ S_2 \] → \[ S_3 \] \( \delta_2, \delta_1, \delta_1^{-1} \) \( \delta_\infty \) \( \delta_1 \)

\[ S_3 \] → \[ S_4 \] \( \delta_1^0 \) \( \delta_1^{-1} \) \( \delta_\infty \) \( \delta_1 \)
Outline

1 Introduction
   • What Is Vectorization
   • What Are The Challenges

2 Structure Of GCC

3 Vectorization

4 Code Generation & Algorithm

5 Summary
for (i=1; i<=100; i++) {
    X[i] = Y[i] + 10; // S1
    for (j=1; j<=100; j++) {
        B[j] = A[j][N]; // S2
        for (k=1; k<=100; k++) {
        }
        Y[i+j] = A[j+1][N] // S4
    }
}

- codegen(R,k,D), R region, k min nesting level, D dependence graph
- Base for gcc’s vectorization procedure
Vectorization Algorithm I

```c
for (i=1; i<=100; i++) {
    X[i] = Y[i] + 10; // S1
    for (j=1; j<=100; j++) {
        B[j] = A[j][N]; // S2
        for (k=1; k<=100; k++) {
        }
        Y[i+j] = A[j+1][N] // S4
    }
}
```

- codegen(R,k,D), R region, k min nesting level, D dependence graph
- Base for gcc’s vectorization procedure
Vectorization Algorithm II
for (i=1; i<=100; i++) {
    codegen({S2,S3,S4},2,D2)
}

X[1:100] = Y[1:100] + 10;
for (i=1; i<=100; i++) {
    for (j=1; j<=100; j++) {
        codegen({S2,S3},3,D3)
    }
    Y[i+1:i+100] = A[2:101][N];
}
X[1:100] = Y[1:100] + 10;
for (i=1; i<=100; i++) {
    for (j=1; j<=100; j++) {
        B[j] = A[j][N];
        A[j+1][1:100] = 
            B[j] + C[j][1:100];
    }
    Y[i+1:i+100] = A[2:101][N];
}
X[1:100] = Y[1:100] + 10;
Auto-Vectorization with GCC

Hanna Franzen, Kevin Neuenfeldt (RWTH)
Outline

1 Introduction
   • What Is Vectorization
   • What Are The Challenges

2 Structure Of GCC

3 Vectorization

4 Code Generation & Algorithm

5 Summary
Overview

- scalar dependence cycles
- data reference dependences
- data reference access patterns
- data reference alignment

Still in progress:
- Cost model
- MIMD support
- Enhancement of existing techniques
- Much more...
Overview

- scalar dependence cycles
- data reference dependences
- data reference access patterns
- data reference alignment

Still in progress:
- Cost model
- MIMD support
- Enhancement of existing techniques
- Much more...
Compile code like:
```
gcc example.c -o example_vect -O3
```
```
/usr/bin/time -f "%E time, %P CPU" ./example
```
0:00.13 time, 97% CPU
```
/usr/bin/time -f "%E time, %P CPU" ./example_vect
```
0:00.03 time, 93% CPU

Speed enhancement of factor \( \approx 4 \Rightarrow VF = 4 \).
Compile code like:
```
gcc example.c -o example_vect -O3
```

- `/usr/bin/time -f "%E time, %P CPU" ./example`
  - 0:00.13 time, 97% CPU
- `/usr/bin/time -f "%E time, %P CPU" ./example_vect`
  - 0:00.03 time, 93% CPU

Speed enhancement of factor $\approx 4 \Rightarrow VF = 4$. 
Compile code like:
gcc example.c -o example_vect -O3

- /usr/bin/time -f "%E time, %P CPU" ./example
  0:00.13 time, 97% CPU
- /usr/bin/time -f "%E time, %P CPU" ./example_vect
  0:00.03 time, 93% CPU

Speed enhancement of factor $\approx 4 \Rightarrow VF = 4$. 
GCC Code

GCC 4.9 Source gcc/tree-vect*

- 31995 well documented LOC
- Introduced on Jan 1st, 2004 in the Ino branch
- Initiated by Dorit Nuzman (IBM)
Multi-platform Auto-vectorization


Auto-vectorization in GCC

Auto-vectorization in GCC - Two years later


Vectorization for SIMD Architectures with Alignment Constraints


Kennedy, Ken and Allen, John R (2001)
Optimizing compilers for modern architectures: a dependence-based approach