Comparação de compiladores C/C++ para processadores x86

Cristopher Carcereri [1] **, Daniel Oliveira** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR) – Curitiba – PR

{clac16,dagoliveira}@inf.ufpr.br

Resumo. *A escolha de compilador pode afetar significativamente a performance de aplicações. Este artigo busca avaliar o impacto de diferentes compiladores C/C++ em aplicações científicas com base no conjunto de benchmarks Rodinia e investigar suas causas. São testados os compiladores AOCC, Clang, GCC e ICC. A comparação mostra que o compilador da Intel apresenta uma vantagem significativa, devida principalmente à sua capacidade de vetorizar o código automaticamente.*

1. Introdução

Com aplicações de alta performance escritas em linguagens de alto nível, a eficiência do código produzido pelo compilador é um fator importante para o uso de recursos e velocidade de processamento atingida.

Entender as diferenças de desempenho entre os códigos gerados é importante tanto para programadores e pesquisadores maximizarem a performance de uma aplicação em uma plataforma de hardware particular, quanto para desenvolvedores de compiladores identificarem oportunidades de melhorar seu produto. Este estudo explora o impacto de quatro compiladores C/C++ amplamente usados — GCC (GNU Compiler Collection), Clang, AOCC (AMD Optimizing C/C++ Compiler) e ICC (Intel oneAPI DPC++/C++ Compiler) — no desempenho de aplicações científicas usando um conjunto abrangente de benchmarks. Além de quantificar o tempo de execução, nos propomos a investigar as causas das disparidades observadas.

[Davis et al. 2021] testou seis compiladores usando cinco aplicações em sistemas heterogêneos com GPUs Nvidia V100. Com a maior parte dos compiladores foi empregado OpenMP para o GPU offload, mas OpenACC e CUDA também foram utilizados. [Halbiniak et al. 2022] examina cinco compiladores para processadores AMD EPYC Rome usando duas implementações, paralelizadas com OpenMP, de modelagem numérica da solidificação de ligas. Ambos os artigos mostraram que o desempenho dos compiladores varia substancialmente.

2. Os testes

Para comparar a performance, foi adotado o conjunto de benchmarks Rodinia (versão 3.1), que é fácil de obter e implementa uma variedade de programas paralelos úteis para aplicações científicas [Che et al. 2009]. Para os testes, foram selecionados 17 (de um total de 19) implementações usando a API OpenMP.[1]

[1]O programa MUMmerGPU foi excluído porque emprega, também, CUDA, não suportado pelas versões mais recentes dos compiladores. Já LU Decomposition mostrou uma imensa sensibilidade ao compilador e as flags, então teria um impacto desproporcional na performance média.

Foram feitas algumas modificações à versão original do Rodinia. Em especial, os makefiles foram alterados para simplificar a compilação com diferentes parâmetros e, para reduzir o impacto de I/O na aferição, a saída dos programas foi direcionada para `/dev/null`. Os benchmarks modificados e os scripts usados para teste, assim como os resultados completos, estão disponíveis em `https://gitlab.c3sl.ufpr.br/clac16/ic-comparacao-compiladores-x86`.

Os compiladores testados foram GCC (14.2.1), Clang (19.1.0), ICC (2025.0.1) e AOCC (5.0). GCC, parte do projeto GNU, inclui front ends para várias linguagens, assim como back ends para diversas arquiteturas. Clang é o front end para linguagens na família C para o LLVM. O Intel C Compiler também é baseado no LLVM, com a versão 2025.0.1 tendo LLVM 19.0.0 como back end. AOCC é um fork do Clang que inclui otimizações da AMD. O AOCC 5.0 tem como back end o LLVM 17.0.6.

Os benchmarks foram compilados e executados em um computador com AMD Ryzen 1700 e 16 GiB de RAM. Para entender o impacto das otimizações, diferentes conjuntos de flags foram testados, mas apenas os mais relevantes serão reportados.[2] Para cada combinação de benchmark, compilador e flags avaliada mediu-se o tempo de oito execuções individuais e calculou-se a média aritmética.

3. Resultados

A Tabela 1 mostra a média geométrica dos tempos médios calculados para os 17 benchmarks com algumas combinações de compilador e flags. Em todos os casos, o desvio padrão geométrico é igual ou inferior a 1.01. São omitidas a flag de dialeto (`-std=gnu89`) usada com alguns programas em C e as flags que habilitam OpenMP (`-qopenmp` com ICC, `-fopenmp` com os demais compiladores).

As diferenças nas flags selecionadas visam compensar variações entre as flags definidas para os compiladores. Com AOCC, Clang e ICC `-O3` implica loop unrolling, mas não com GCC. E, por default, ICC usa um modelo de ponto flutuante (`-fp-model=fast=1`) que habilita otimizações que sacrificam a acurácia da aritmética, ao contrário dos demais. O comportamento default destes compiladores foi equiparado com a flag `-fp-model=precise` do ICC.

Compilador e flags	**Tempo de execução (Média Geométrica)**
aocc -O3	47.28
aocc -O3 -march=native	48.51
aocc -O3 -march=native -fno-tree-vectorize -fno-tree-slp-vectorize	49.13
clang -O3	46.7
clang -O3 -march=native	47.75
clang -O3 -march=native -fno-tree-vectorize -fno-tree-slp-vectorize	48.1
gcc -O3 -funroll-loops	45.92
gcc -O3 -funroll-loops -march=native	44.69
gcc -O3 -funroll-loops -march=native -fno-tree-vectorize -fno-tree-loop-vectorize -fno-tree-slp-vectorize	45.06
icx -O3 -fp-model=precise	43.37
icx -O3 -march=native -fp-model=precise	42.7
icx -O3 -fma -march=core-avx2 -fp-model=precise -no-vec	44.47

Tabela 1. Tempos de execução médios para diferentes parâmetros de compilação

[2]Os dados completos estão disponíveis em `https://gitlab.c3sl.ufpr.br/clac16/ic-comparacao-compiladores-x86`.

O compilador da Intel obteve o maior desempenho. Comparando o melhor caso na tabela para cada compilador, ICC superou o GCC em cerca de 5%, e GCC superou o Clang em cerca de 4%. Para todos os conjuntos de flags testados, a performance média do Clang supera a do AOCC entre 1% e 3%.

Com a flag `-march=native` há uma melhora sensível nos tempos de execução para GCC e ICC, enquanto há uma piora para Clang e AOCC, indicando que os dois têm problemas ao tentar utilizar extensões como AVX2.

4. Análise dos códigos

Para investigar as causas das diferenças observadas, analisamos o código do programa em que o compilador da Intel obteve maior vantagem, Particle Filter. O assembly foi gerado com as flags `-O3` e `-march=native`, além de `-funroll-loops` com GCC e `-fp-model=precise` com ICC. Com estas flags, ICC mostrou um desempenho 42% superior ao do GCC, que, por sua vez, foi 24% mais rápido que Clang e AOCC.

A Listagem 1 contém um segmento de Particle Filter. Os quatro compiladores o vetorizam. No entanto, GCC usa registradores de 128 bits (2 doubles), enquanto os demais usam registradores de 256 bits (4 doubles). A cada iteração, GCC opera sobre 8 vetores (16 elementos) e AOCC, Clang e ICC operam sobre 1 vetor (4 elementos).

Listagem 1. Segmento de Particle Filter (linhas 445–447)

```
for(x = 0; x < Nparticles; x++){
  weights[x] = weights[x]/sumWeights;
}
```

As observações feitas para a Listagem 1 se aplicam a quase todo segmento de Particle Filter analisado. GCC é o mais agressivo no loop unrolling e, ao vetorizar, sempre se limita ao uso de registradores XMM, de 128 bits. De acordo com o tamanho do laço, é usado um fator de 4, 8 ou 16, com o caso típico sendo 8 elementos ou, em código vetorizado, 8 vetores (16 elementos). Ao vetorizar o código, ICC geralmente opera sobre 1 vetor (4 elementos) por iteração. Em código não vetorizado, opera sobre 4 ou 8 elementos por iteração. Clang e AOCC geram códigos bastante parecidos entre si e geralmente operam sobre um único elemento ou vetor por iteração.

Seguindo o padrão observado, AOCC e Clang geram, para o segmento na Listagem 2, um código praticamente idêntico e sem loop unrolling. O laço principal consiste de 6 instruções, incluindo 1 leitura da memória, 1 comparação ponto flutuante e 2 saltos condicionais. Com GCC, o laço principal compara 8 elementos por iteração e consiste de 35 instruções, incluindo 8 leituras da memória, 8 comparações ponto flutuante e 9 saltos condicionais. ICC é o único que vetoriza o fragmento. O laço principal realiza 16 comparações e consiste de 12 instruções, incluindo 4 comparações de (vetores) ponto flutuante lidos da memória e 2 saltos condicionais.

Listagem 2. Segmento de Particle Filter (linhas 288–293)

```
for(x = 0; x < lengthCDF; x++) {
  if(CDF[x] >= value) {
    index = x;
    break;
```

```
    }
  }
```

O segmento na Listagem 2 é a principal parte da seção que domina o tempo de execução do programa e que executa aproximadamente 2.4 vezes mais rápido no programa gerado pelo ICC do que no produzido pelo GCC, e cerca de 1.4 vezes mais rápido no GCC do que no Clang e AOCC.

O programa para o qual ICC apresenta a segunda maior vantagem, SRAD, revela uma situação similar. Tomando-se as mesmas flags que a análise anterior, o desempenho do ICC é cerca de 36% melhor que o do Clang e do AOCC, e a diferença se deve a dois laços que apenas o compilador da Intel vetoriza (resultando em execuções 1.6 vezes mais rápidas para um laço e 2.3 vezes mais rápidas para o outro). Naturalmente, outros fatores impactam o desempenho. Apesar de GCC, Clang e AOCC nem vetorizarem nem realizarem unroll em um desses laços, a versão do GCC o executa em um tempo 2.7 vezes maior que a dos outros dois, contribuindo para uma performance 49% pior que a desses compiladores.

5. Conclusão

A análise demonstra que o compilador da Intel identifica possibilidades de vetorização que os outros compiladores ignoram. GCC, em especial, mesmo ao vetorizar um código, pode não ser capaz de utilizar toda a largura dos registradores.

De fato, como mostra a Tabela 1, ICC é o compilador mais afetado ao se desabilitar a vetorização automática, com sua performance média piorando em cerca de 4%. A vantagem frente ao GCC é reduzida para cerca de 1%. Mas a análise do código revela que, mesmo com a flag `-no-vec`, ICC ainda mantém vetorizações mais simples, incluindo a correspondente ao segmento na Listagem 1, em que não apenas a vetorização é mantida, mas o compilador passa a realizar loop unrolling para operar sobre dois vetores (oito elementos) por iteração. Portanto, a redução de desempenho observada não reflete totalmente o efeito da vetorização para o ICC.

Isto sugere que, enquanto várias diferenças nas estratégias de otimização afetam o resultado, a principal vantagem do ICC é sua capacidade de vetorização.

Referências

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pages 44–54. Ieee.

Davis, J. H., Daley, C., Pophale, S., Huber, T., Chandrasekaran, S., and Wright, N. J. (2021). Performance assessment of openmp compilers targeting nvidia v100 gpus. In *Accelerator Programming Using Directives: 7th International Workshop, WACCPD 2020, Virtual Event, November 20, 2020, Proceedings 7*, pages 25–44. Springer.

Halbiniak, K., Wyrzykowski, R., Szustak, L., Kulawik, A., Meyer, N., and Gepner, P. (2022). Performance exploration of various c/c++ compilers for amd epyc processors in numerical modeling of solidification. *Advances in Engineering Software*, 166:103078.