GNU Compiler Collection 4.2

Correndo em paralelo

A última versão do compilador do projeto GNU oferece maior suporte à programação paralela, além de alguns recursos de otimização. Confira nosso test-drive.

por René Rebe e Susanne Klaus

Depois de muito debate e o atraso de praxe, a versão mais recente do compilador *C*, *C++*, *Objective-C*, *Fortran*, *Ada* e *Java* do Projeto GNU finalmente tomou forma. A versão 4.2 do GCC[1] segue a tendência de várias mudanças maiores e menores. A lista completa das modificações se encontra na página do compilador[2].

A mudança mais significativa na versão 4.2 é o suporte ao *OpenMP*[3], um padrão aberto para a paralelização de programas — especialmente para sistemas com memória compartilhada. O OpenMP permite que os programadores especifiquem como o compilador e os sistemas de tempo de execução devem distribuir os segmentos de código ao longo de múltiplas *threads* para execução paralela em sistemas com mais de um núcleo.

A implementação do OpenMP na nova versão do GCC simplifica a programação paralela nos sistemas suportados. Os desenvolvedores não precisam mais seguir o processo complicado e propenso a erros de personalizar seu código-fonte de acordo com as *APIs* específicas de determinadas arquiteturas.

Recursos do OpenMP

A versão atual do GCC suporta todos os recursos do OpenMP. Durante a compilação, as palavras-chave determinam como a execução paralela do código é efetuada.

Os *pragmas* permitem que os compiladores gerem código com as extensões do OpenMP sem nem terem ciência de que estão sendo tratados por esse sistema, o que evita códigos específicos por plataforma e uma selva impenetrável de `ifdef`. A palavra-chave `#pragma omp ...` faz o compilador realizar a otimização paralela. Por baixo dos panos, o GCC utiliza a biblioteca *Pthread* para criar as threads em sistemas Unix e semelhantes, como o Linux.

No caso mais simples, loops com um espaço de iteração significativo — ou seja, sempre que se ganhar algo criando novas threads — são marcados da seguinte forma:

```
#pragma omp for
for(i=0; i<N; i++)
  a[i] += b[i];;
```

O OpenMP possui controles flexíveis; por exemplo, pode-se mandar uma thread em um algoritmo complexo utilizar uma variável local, a qual o programa adiciona ao final (redução):

```
#pragma omp parallel for \
private(w) reduction(+:sum) \
schedule(statuc,1)
for(i=0; i<N; i++) {
  w = i*i;
  sum = sum + w*a[i];
}
```

A possibilidade de se certificar da ID da thread é mais útil para testes do que para algoritmos:

```
#pragma omp parallel private(id)
int id=omp_get_thread_num();
printf("Sou a thread %d\n",id);
```

Com o crescimento contínuo da família de processadores *x86*, temos agora duas novas opções de arquitetura x86. A diretiva de arquitetura `native` diz ao GCC para aplicar a melhor otimização possível para o processo existente em tempo de compilação, com base nas instruções *cpuid*, enquanto `generic` gera programas que sempre rodarão igualmente bem em CPUs AMD, Intel ou Via.

Alertas e adições

Um novo alerta, que pode ser ativado com a opção `-Waddress` e já está contido em `-Wall`, aponta erros típicos de programação que ocorrem em comparações entre endereços de literais de *strings* e ponteiros para funções. A opção `-Wextra` emite um alerta no caso de uma expressão `if` vir seguida de um ponto-e-vírgula, para evitar erros de digitação como esse:

```
if(a);
return 1;
return 0;
```

O novo compilador promete reduzir o tempo de início dos programas e, mais especificamente, a espera para o *linker* dinâmico resolver os símbolos – um problema do qual os desenvolvedores já se queixam há certo tempo, principalmente no

	Bzip2	Gnupg	Gzip	Lame	OpenSSL	Tramp3d
3.4.0 -00	1.73	16.20	0.92	14.12	63.76	17.76
4.0.0 -00	1.73	15.30	0.91	13.88	59.34	15.54
4.1.0 -00	1.69	15.14	0.93	13.66	59.03	16.21
4.2.0 -00	1.70	14.76	0.84	13.81	58.65	15.16
3.4.0 -01	2.06	20.96	1.29	17.36	73.69	33.99
4.0.0 -01	2.45	22.04	1.59	18.21	74.22	64.11
4.1.0 -01	2.80	22.34	1.58	19.43	76.53	51.70
4.2.0 -01	3.16	25.12	1.77	20.18	79.92	45.10
3.4.0 -0s	3.10	25.98	1.50	20.40	82.20	45.18
4.0.0 -0s	2.67	25.02	1.66	19.32	79.69	24.53
4.1.0 -0s	2.92	26.07	1.80	20.54	82.84	26.85
4.2.0 -0s	3.32	28.80	1.93	20.93	86.65	30.41
3.4.0 -02	3.38	27.20	1.71	20.77	87.22	47.95
4.0.0 -02	3.49	26.84	1.90	22.48	85.97	79.83
4.1.0 -02	3.77	27.96	2.05	23.34	88.19	64.67
4.2.0 -02	4.46	31.41	2.15	24.89	94.54	58.52
4.1.0 -02 -loops	4.38	31.66	2.69	29.29	93.19	68.42
4.2.0 -02 -loops	5.15	34.92	2.79	30.57	100.17	65.09
4.1.0 -02 -tracer	3.81	28.63	1.94	23.84	88.44	65.91
4.2.0 -02 -tracer	4.48	31.85	2.22	25.08	95.54	61.23
4.1.0 -02 -vect	3.94	28.63	2.10	24.12	89.05	64.14
4.2.0 -02 -vect	4.58	32.02	2.26	25.48	95.31	58.95
4.1.0 -02 -x	5.10	37.35	2.92	32.91	97.23	72.26
4.2.0 -02 -x	6.07	40.94	3.28	34.05	104.21	65.00
3.4.0 -03	3.98	29.80	1.96	22.36	91.03	49.71
4.0.0 -03	3.92	28.64	2.15	23.80	87.60	87.50
4.1.0 -03	4.21	30.45	2.35	26.86	92.29	71.94
4.2.0 -03	5.14	34.76	2.63	28.29	98.69	61.40
4.1.0 -03 -loops	4.82	34.32	3.00	31.24	97.54	73.01
4.2.0 -03 -loops	5.70	38.56	3.21	33.43	105.38	64.34
4.1.0 -03 -tracer	4.39	30.73	2.43	27.59	91.71	70.75
4.2.0 -03 -tracer	5.21	34.81	2.67	28.57	99.22	63.09
4.1.0 -03 -vect	4.47	31.79	2.42	27.55	92.11	69.21
4.2.0 -03 -vect	5.35	35.73	2.71	29.07	99.16	61.23

(in seconds – smaller is better)

Figura 1 Resultados do *benchmark* para tempos de compilação de programas no teste.

caso do C++. Símbolos locais não são mais visíveis por padrão, e o compilador aplica automaticamente atributos `visibility` de classes a seus membros.

A opção `-fno-toplevel-reorder` agora possibilita a exibição de funções e variáveis na ordem em que aparecem no código-fonte, para códigos como um assembler *inline*, que dependem de uma ordem específica no código.

Deve-se notar que ocorrem novas otimizações de *overflow* ao se empregar o nível de otimização `-O2`. O novo compilador pode presumir que um overflow não vai ocorrer num loop como `for(int i=1;i>0;*i=2)`, e assim otimiza o resultado para formar um loop infinito.

Os desenvolvedores do GCC adicionaram novas funcionalidades ao novo padrão *200x* do C++, que ainda está na fase de padronização. Por exemplo, o *namespace* `TR1` agora inclui `<random>` e `<complex>`. Os *templates* de contêineres livres de *locks* desenvolvidos durante o Google Summer of Code também foram integrados.

Regressão

A boa notícia é que a versão 4.2 do GCC não introduz muitas novas falhas. Um curto teste, no qual usamos o novo compilador para compilar um sistema T2[4] completo, produziu apenas dois erros.

Entretanto, foi necessário muito mais memória para compilar alguns arquivos do pacote do servidor *Xorg*[5], forçando o kernel a terminar o compilador em sistemas com menos de 1 GB de memória. Além disso, o *OpenSSL* utiliza *typecasts* de ponteiros de funções[6] de uma forma que o padrão C não define; isso faz o programa fechar em tempo de execução[7].

Benchmarks

A máquina do laboratório era um Intel Core 2 Duo 2 GHz com 1 GB de RAM. Utilizamos a versão atual do *Open Bench* para testar as versões 3.4, 4.0, 4.1 e 4.2 do GCC, compilando com o processador em modo de 64 bits para programas de 64 bits. Medimos os tempos de compilação (**figura 1**) e de execução em segundos (ou o de execução por iteração em milissegundos, no caso do OpenSSL) (**figura 2**).

Tempos

Numa primeira verificação, percebemos que a versão 4.2 gasta mais tempo realizando otimizações do que seus antecessores. A recompensa por esse esforço é um menor tempo de execução, mesmo em programas C legados.

	Bzip2	Gnupg	Gzip	Lame	OpenSSL	Tramp3d
3.4.0 -00	20.39	17.01	16.68	149.90	4.81	177.56
4.0.0 -00	20.86	17.34	16.20	145.69	4.91	160.82
4.1.0 -00	20.50	20.00	16.17	144.28	4.79	160.81
4.2.0 -00	20.30	17.73	16.18	142.15	4.78	160.22
3.4.0 -01	8.54	9.22	7.73	69.55	1.26	13.85
4.0.0 -01	8.86	9.77	7.78	67.95	1.23	5.36
4.1.0 -01	9.13	8.57	7.65	66.93	1.21	4.29
4.2.0 -01	9.30	8.87	8.06	66.07	1.15	4.78
3.4.0 -0s	8.59	9.35	7.46	66.44	1.27	8.95
4.0.0 -0s	9.06	11.29	8.28	66.41	1.21	44.67
4.1.0 -0s	9.22	9.12	8.22	64.74	1.24	10.35
4.2.0 -0s	8.57	8.98	8.70	64.54	1.18	10.47
3.4.0 -02	8.54	10.79	7.46	66.54	1.22	8.64
4.0.0 -02	8.62	9.74	7.21	64.88	1.19	5.21
4.1.0 -02	9.10	8.86	7.23	64.63	1.18	3.96
4.2.0 -02	8.70	6.02	7.99	62.81	1.14	4.64
4.1.0 -02 -loops	8.86	8.88	7.34	56.29	1.19	2.59
4.2.0 -02 -loops	8.49	8.96	7.99	56.72	1.15	3.38
4.1.0 -02 -tracer	9.09	8.86	7.34	64.93	1.20	3.95
4.2.0 -02 -tracer	8.66	8.98	7.88	62.18	1.17	4.62
4.1.0 -02 -vect	9.09	8.42	7.26	64.81	1.21	3.98
4.2.0 -02 -vect	8.71	8.35	8.06	62.72	1.13	4.61
4.1.0 -02 -x	8.80	8.29	7.34	56.49	1.21	2.60
4.2.0 -02 -x	8.49	8.69	7.91	56.36	1.18	3.39
3.4.0 -03	8.19	9.26	7.51	61.41	1.18	8.63
4.0.0 -03	8.71	9.57	7.30	64.75	1.18	5.19
4.1.0 -03	9.07	8.97	7.27	64.65	1.19	2.80
4.2.0 -03	8.59	9.19	8.01	62.05	1.13	3.61
4.1.0 -03 -loops	8.79	8.34	7.22	56.16	1.18	2.58
4.2.0 -03 -loops	8.59	8.88	8.02	56.70	1.14	3.46
4.1.0 -03 -tracer	8.98	8.41	7.34	64.56	1.19	2.82
4.2.0 -03 -tracer	8.62	9.29	7.84	62.09	1.17	3.60
4.1.0 -03 -vect	9.03	9.39	7.21	64.48	1.19	2.98
4.2.0 -03 -vect	8.70	9.23	8.00	62.51	1.13	3.60

(in seconds – smaller is better)

Figura 2 Esse diagrama mostra os tempos de execução para várias versões do compilador.

Embora o compilador seja bem mais lento quando os níveis de otimização `-O2` ou `-O3` são ativados, o tempo de compilação durante o desenvolvimento do software com `-O0` é menor.

Uma rápida inspeção dos logs do nosso benchmark revela que o novo compilador vetoriza mais loops — 14 para o *Gzip*, contra 12 no GCC 4.1.

Conclusões

A integração do OpenMP ao GCC 4.2 facilita a tarefa de programar para sistemas com múltiplos núcleos. Isso ajuda o compilador livre a acompanhar as alternativas proprietárias.

Graças à introdução já bastante difundida de processadores multi-núcleo, a parelização se tornou um importante tópico para muitos programadores. O fato de que cada versão do compilador se dedicou mais a otimizar o código é um pouco preocupante.

Novos projetos

Os novos projetos agendados para serem finalizados antes de sair a nova versão 4.3 do GCC incluem o compilador Java do projeto *Eclipse*, que oferece total suporte à integração do Java 1.5. A integração da biblioteca *MPFR* ajudará a padronizar chamadas a funções matemáticas padrão.

O suporte ao futuro padrão 200x do C++ será ampliado na próxima versão do GCC. As funções de otimização para tipos de processador mais recentes, como o Core 2 Duo e o AMD Geode, já se encontram na atual versão de desenvolvimento do GCC. ■

Mais Informações

[1] Página do GCC: http://gcc.gnu.org/

[2] Changelog do GCC 4.2: http://gcc.gnu.org/gcc-4.2/changes.html

[3] OpenMP: http://www.openmp.org/

[4] T2 SDE: http://www.t2-project.org/

[5] Relato de falha no servidor Xorg: http://tinyurl.com/2ft4az

[6] Patch para o OpenSSL com GCC 4.2: http://tinyurl.com/22tjxq

[7] Open Bench: http://tinyurl.com/2ldgag

Os autores

René Rebe e **Susanne Klaus** são diretores da Exact CODE, na Alemanha, e envolvem-se com diversos projetos de Código Aberto em seu trabalho diário.