Avaliação de desempenho na execução de aplicações

TPC9 e Guião laboratorial

Alberto José Proença

Objectivo

A lista de exercícios/tarefas propostos no TPC9 / Guião como objectivos:

- a aquisição de **técnicas de medida de tempos de execução** (microscópicos) **de programas elaborados em C**, e respectiva compreensão/ interpretação de resultados, ao nível do *assembly*;
- a realização de **medições de desempenho das variantes de optimização dum dado programa** (evoluções para programas mais eficientes), e posterior análise de algumas das técnicas de optimização utilizadas.

Os exercícios para serem resolvidos antes da aula TP estão assinalados com uma caixa cinza.

Contexto

Transformação sistemática de um programa noutro mais eficiente

O programa a estudar é o exemplo de optimização analisado no texto de apoio (de Bryant) na secção 5.3: parte de uma simples estrutura de um vector de dados – *Vector Abstract Data Type*, adiante designado por vector ADT – para combinar todos os elementos dum vector num único valor, de acordo com uma dada operação. O conjunto de programas em C relevantes para este trabalho, apresentados no livro nos capítulos 5 e 7, são disponibilizados em formato electrónico, de modo compactado, junto a este enunciado.

O vector ADT permite a definição de vários tipos de dados como elementos constituintes do vector; neste exercício, iremos apenas considerar **o tipo `int`**. Os ficheiros relacionados com o vector ADT são o `vec.h` e o `vec.c`. Mais detalhes no texto de apoio (em 5.3).

O exemplo de optimização – que combina os valores dum vector – encontra-se nos ficheiros `combine.h` e `combine.c`. Usando diferentes definições das constantes `IDENT` e `OPER`, o código pode ser recompilado para efectuar operações distintas aplicadas aos dados. Iremos apenas considerar **a operação de soma**. Mais detalhes no texto de apoio (em 5.3).

Técnicas de medição

Uma das melhores formas de exprimir o **desempenho da execução de um programa** que efectua essencialmente um cálculo repetitivo - normalmente sob a forma de um ciclo – é utilizando uma métrica que relacione o tempo necessário para realizar o conjunto de operações que se efectua sobre cada um dos elementos da lista (repetitiva), com uma característica temporal do CPU, tornando a métrica independente da frequência do *clock* do CPU. Assim, a métrica que iremos utilizar é baseada na proposta no texto de apoio em 5.2: o número médio de **ciclos de *clock* por elemento** processado **(CPE)**.

Uma **forma simplificada para se obter o valor de CPE** (mas pouco precisa), consiste em **(i)** inicializar um "cronómetro", **(ii)** executar o programa de teste (a medir), **(iii)** contabilizar o número de ciclos de *clock* desde a última inicialização do contador e **(iv)** calcular o respectivo valor por

elemento, dividindo o nº total de ciclos pelo nº de elementos da lista.

Para efectuar **medições de tempos "microscópicos" com grande precisão**, alguns processadores das últimas gerações disponibilizam facilidades extras com este objectivo. Contudo, a precisão dos resultados pode ser influenciada por diversos factores do sistema de computação, conforme referido em 9.4.1 e 9.4.2:

- **o nº de processos que simultaneamente partilham o CPU** durante a execução do programa em estudo; para reduzir/eliminar a influência deste factor, este caso de estudo utiliza um programa com um tempo de execução pequeno (< 7 mseg), garantindo assim uma muito baixa probabilidade de interrupção (i.e, baixa probabilidade de ocorrência de um *context switching*); adicionalmente fazem-se 5 *runs* e selecciona-se o melhor tempo;
- **os diferentes tempos de accesso aos diversos níveis da hierarquia de memória;** para minimizar este factor, este caso de estudo foi dimensionado para caber todo na *cache* mais rápida (1º nível), e são disponibilizadas rotinas para "aquecer" a *cache* antes da medição dos tempos de execução (disponíveis em `time_p.c`)

No caso concreto do IA32 (apenas após o Pentium), **a Intel disponibiliza um contador de 64 bits** (o *cycle counter*) **e uma nova instrução, `rdtsc`** (de "*read time stamp counter*"). Os programas em C que permitem o manuseamento deste contador como "cronómetro" podem ser analisados em `clock.h` e `clock.c`. Mais detalhes na secção 9.3 do texto de apoio.

Uma análise deste código (também disponível na Fig. 9.9 do texto de apoio) mostra a **inserção de código em *assembly* directamente no ficheiro com o programa em C** (característica não normalizada de acordo com ANSI C). A inserção de código *assembly* em linha é tratado no texto de apoio em 3.15, onde em 3.15.2 se mostra, de maneira simples, como se faz a utilização do comando `asm`, reconhecido pelo `gcc`, com uma sintaxe extremamente simples:

`asm( code-string [ : output-list [ : input-list [ : overwrite-list ]]]);`

Exercícios

1. Analise o código disponibilizado em `combine.c` e `vec.c`. Onde encontrar `IDENT` assuma "`0`", e `OPER` assuma "`+`".
A partir das funções existentes, escreva um programa em C para calcular o CPE (forma simplificada, ver em cima) da função `combine1`, apenas para a operação soma de inteiros.
Condições de teste: vector com 1024 elementos, atribuindo a cada elemento o valor 32.
Sugestão para o `main`:

```
int main()
{
    int i, dimensao = 1024;
    vec_ptr vector = new_vec(dimensao);
    data_t resultado;

    double cpe;

    /*Inicialização do vector*/
    for(i=0; i<dimensao; i++){
      set_vec_element(vector, i, 32);
    }
    combine1(vector, &resultado);    /* Aquecimento da cache */

    start_counter();
    combine1(vector, &resultado);
    cpe = get_counter();
    cpe = cpe / dimensao;
    printf("CPE a quente (%s, %s, %d elem): %.2f\n", OPER_NAME, DATA_NAME, dimensao, cpe);
}
```

2. Faça a compilação sem qualquer optimização.
Teste num computador à sua escolha, executando 5 vezes cada o programa e considerando os melhores valores de CPE obtidos. Anote os resultados obtidos.
Leve o programa para a aula para fazer medições num computador do laboratório.
Comente resultados díspares que encontrar, e tente arranjar uma justificação plausível.

3. As restantes funções em C (disponibilizado em formato electrónico conjuntamente com este enunciado) são variantes de optimização da função original `combine1`.
Usando como "*template*" a resolução exercício **1**, escreva programas em C para cada uma das funções apresentadas.
Alternativamente, escreva apenas um `main` que inclua as chamadas das diversas funções de optimização.

Sugestão:

```
/* Execucao de "combine1" com a cache a frio */
    fprintf(f, "Combine1\n");

   for (i = 0; i < 5; i++) {
    clear_cache();

    start_counter();
    combine1(vector, &resultado);
    cpe = get_counter();
    cpe = cpe / dimensao;
    fprintf(f, "CPE a frio (%s, %s, %d elem): %.2f\n", OPER_NAME, DATA_NAME,
dimensao, cpe);
   }

   /* Execucao de "combine1" com cache quente*/

   for (i = 0; i < 5; i++) {
    start_counter();
    combine1(vector, &resultado);
    cpe = get_counter();
    cpe = cpe / dimensao;
    fprintf(f, "CPE a quente (%s, %s, %d elem): %.2f\n", OPER_NAME,
DATA_NAME, dimensao, cpe);
   }
```

a) Compile com optimização `-O2` esses programas, e execute 5 vezes cada um deles; considere os melhores valores de CPE obtidos para cada versão optimizada, e preencha a coluna apropriada da tabela fornecida.

b) Identifique (na tabela fornecida) os tipos de optimização efectuados (se dependentes ou não da máquina) e caracterize os métodos usados.

c) Considere apenas as optimizações independentes da máquina. Identifique ao nível do *assembly* as optimizações introduzidas.

Tempos de execução de `combine` e variantes

Cálculo de CPE's (versão simplificada)

Função	**Método**	**Inteiro** *soma*
`combine1`	ADT, não optimizado	
`combine1`	ADT, `-O2`	
`combine2`		
`combine3`		
`combine4`		
`combine5`		
`combine6`		
`unroll4x2`		

Identificação das optimizações ao nível do *assembly*: