SUGGESTING LOOP UNROLLING USING A HEURISTIC-GUIDED APPROACH

Pedro Pinto

Dissertaçao realizada sobre a orientaçao do *Prof. João Cardoso*
na *FEUP*

1. Motivação

O desenvolvimento de aplicações embebidas encontra frequentemente restrições de memória e/ou de tempo de execução e, como tal, o desempenho deve ser considerado durante o processo de desenvolvimento. Uma maneira de melhorar o desempenho é usando transformações de código, algo que já é feito atualmente por compiladores e ferramentas de otimização. No entanto, há casos em que essas transformações não podem ser automaticamente aplicadas e cabe ao programador aplicá-las manualmente.

É necessário uma ferramenta que possa ajudar os programadores a tomar decisões informadas sobre quando e como aplicar estas transformações, sabendo que vão levar a uma melhoria de desempenho. Como é o programador que irá fazer a transformação, esta ferramenta deve conseguir ajudar mas mantendo o programador no controlo. Uma possível solução é fazer sugestões sobre que transformações seriam beneficiais, dando uma indicação da direção a seguir.

2. Objetivos

O objetivo principal desta dissertação é propor uma abordagem para ajudar os programadores a decidir sobre a aplicação do *Loop Unolling* e sobre o fator de *unroll* a usar. Esta abordagem irá usar um conjunto de heurísticas e informação do código fonte para prever se o *Loop Unrolling* irá trazer uma melhoria de desempenho.

Esta abordagem será usada por um programador, logo, faz sentido trabalhar num nível elevado e usar transformações de código fonte. *Loop Unrolling* é uma boa transformação, porque já foi extensamente estudada e vem sendo usada há décadas. A sua aplicação é fácil e sempre legal.

Esta abordagem deve ser traduzida para um protótipo de modo a que possa ser validada e avaliada. As heurísticas encontradas devem ser instanciadas com foco numa arquitetura específica, que seja significativa no contexto das aplicações e sistemas embebidos.

3. Descrição do Trabalho

Inicialmente é apresentada a abordagem proposta e as heurísticas utilizadas nessa abordagem. Depois, é apresentado o protótipo que foi desenvolvido e, finalmente, são apresentados os resultados dos testes e a sua análise.

3.1. Abordagem Proposta

A abordagem consiste num processo de quatro etapas. Inicialmente, o *loop* a analisar é convertido para um modelo que mantém a informação relevante. Este modelo é usado ao longo de todos os passos e precisa de informações sobre os limites de iteração, o *step* e a variável de indução. Também precisa de saber o número de instruções no corpo do *loop* e se são feitos acessos a *arrays*.

O segundo passo utiliza o modelo para testar o *loop* usando um conjunto de regras de aceitação. O principal objetivo dessas regras é garantir que o número de iterações é conhecido em tempo de compilação. Se isso não for possível, este teste falhará e o *loop* é descartado.

Se o *loop* respeita as regras anteriores pode então ser avaliado, que é o terceiro passo. Os *loops* têm uma pontuação que inicialmente é 0. Esta pontuação vai mudando à medida que o *loop* é avaliado pelas heurísticas. Se, no fim da avaliação, a pontuação está acima de um limiar, o *loop* é considerado um bom candidato e irá avançar para o último passo.

Neste ponto, a única coisa que resta a fazer é escolher um fator de *unroll* adequado. O *loop* já foi testado duas vezes, portanto, assume-se que ele irá beneficiar da transformação. Isto permite usar uma estratégia muito simples, podemos escolher o maior fator que não cause *thrashing* da *cache* de instruções.

3.2. Heurísticas

Há cinco heurísticas diferentes na abordagem proposta. As primeiras quatro procuram características que, quando encontradas, é provável que conduzam a uma melhoria de desempenho.

Small Iteration Count Esta heurística recompensa *loops* que executam um número de iterações pequeno. Um *loop* tem um número pequeno de iterações, é menos suscetível a causar *thrashing* da *cache* de instruções. Há também pouco perigo ao aplicar *Unrolling* total.

Data Reuse Esta heurística procura oportunidades para reutilizar dados. Se um *loop* partilha dados ao longo de várias iterações, o *Loop Unrolling* pode, ao expor essas iterações, permitir que o compilador reutilize esses dados. A distância entre iterações que utilizam os mesmos dados é chamada distância reutilização. Esta heurística dá uma pontuação maior

a distâncias mais curtas porque estas permitem a reutilização de mais valores com o mesmo fator de *unroll*.

Same Scope Array Se o *loop* itera sobre um *array* de constantes e é possível ver a sua declaração, o *Loop Unrolling* pode permitir a substituição dos acessos ao *array* pelos próprios valores. Se um tal *array* existe, depois de aplicar *Loop Unrolling* total e transformações como *Constant Propagation* e *Constant Folding*, o compilador pode remover os acessos ao *array* e substituí-los com os valores constantes.

Loop Body Execution Time Relation Uma das principais vantagens do *Loop Unrolling* é que, independentemente de tudo o mais, é possível reduzir a sobrecarga de estrutura de controlo. Cada iteração executa uma estrutura de controlo, responsável por atualizar a variável de indução e sair corretamente do *loop*. Esta transformação reduz o número de iterações, o que por sua vez reduz o tempo gasto na execução deste código de controlo.

Number of Instructions O *Loop Unrolling* aumenta o número de instruções no interior do corpo do *loop*, o que pode ter um efeito negativo no desempenho da *cache* de instruções. Quanto maior for o fator de *unroll*, mais instruções existem no corpo do *loop*. Esta situação pode conduzir a um aumento do número de falhas na *cache*, o que resultará numa perda de desempenho que pode ofuscar quaisquer benefícios da transformação.

3.3. Protótipo

Foi desenvolvido um protótipo direcionado para a arquitetura PowerPC, e mais especificamente para o processador PowerPC 604. As heurísticas e as métricas utilizadas para sugerir *Loop Unrolling* e o fator de *unroll* foram instanciadas para esse processador. Através da observação e experimentação empírica, foi possível encontrar valores adequados para o processador alvo.

O protótipo usa uma infraestrutura de compiladores *source-to-source*, o Cetus. A principal tarefa do Cetus é gerar uma Árvore de Sintaxe Abstrata (ASA) a partir do código fonte. Esta ASA é usada para criar, para cada *loop* candidato, um modelo que é passado a um motor de avaliação. Este motor tem pleno conhecimento das heurísticas e seus valores e consegue avaliar o *loop*.

O Cetus é usado mais uma vez para mostrar ao programador a sugestão resultante da avaliação. Ao transformar a ASA, é possível criar um comentário com a sugestão antes de cada *loop* analisado. O Cetus irá imprimir o conteúdo da ASA para um ficheiro de código fonte, onde o programador pode ver o resultado da avaliação.

3.4. Resultados

De uma forma geral, os resultados são positivos. Dos 8 testes avaliados, foram feitas sugestões corretas para 6. A Tab. 1 mostra a avaliação de cada teste. É possível ver o resultado da avaliação, a sugestão (aplicar *Unrolling* ou manter) e se esta sugestão é correta.

Tab. 1 – As sugestões feitas para cada teste.

Teste	Resultado	Sugestão	Correta
Dot Product	2	Aplicar	Sim
Gouraud	-8	Manter	Não
Grid Iterate	-10	Manter	Sim
Vector Sum	2	Aplicar	Sim
ISO1	6	Aplicar	Sim
ISO2	20	Aplicar	Sim
FSD1	10	Aplicar	Sim
FSD2	-6	Manter	Não

A Tab. 2 mostra a sugestão do fator de *unroll*. É possível ver, para cada teste sugerido para o *Unrolling*, o fator de *unroll* sugerido e a melhoria de desempenho associada. Para fins de comparação, também é possível ver o fator de *unroll* ótimo, isto é, o fator de *unroll* que conduz à maior melhoria de desempenho. Em 4 dos 5 testes, o fator de *unroll* é igual ou próximo do fator ótimo e no outro teste há uma diferença de desempenho de apenas 3,88%.

Tab. 2 – Comparação do fator de *unroll* sugerido com o fator de *unroll* ótimo (e as melhorias de desempenho associadas) para os testes que foram classificados bons candidatos.

	Fator de *Unroll*		Melhoria	
Teste	Sugerido	Ótimo	Sugerido	Ótimo
Dot Product	57	58	37,25%	37,27%
Vector Sum	57	253	31,49%	35,37%
ISO1	3	3	33,30%	33,30%
ISO2	3	3	78,62%	78,62%
FSD1	8	8	6,33%	6,33%

4. Conclusões

Esta dissertação apresenta uma abordagem diferente para o problema da otimização de desempenho, usando heurísticas, uma transformação de código fonte e sugestões. Os resultados mostram que a abordagem pode ajudar o programador, indicando que *loops* devem ser transformados e sugerindo fatores de *unroll* adequados. Mesmo que os seus valores sejam sempre adaptados para uma arquitetura específica, as heurísticas apresentadas podem ser usadas para qualquer arquitetura porque dependem principalmente de características presentes no código.

Esta abordagem tem algumas limitações. Ela considera apenas *loops FOR* interiores cujo número de iterações é conhecido em tempo de compilação. Portanto, analisa apenas uma pequena parte de todos os possíveis *loops*. Além disso, a estratégia utilizada para escolher um fator de *unroll* poderia melhorada, pois apenas toma em consideração o número de instruções no corpo do *loop* e seu efeito na *cache*. A redução da sobrecarga, uma das principais vantagens desta transformação, não é devidamente contabilizada, embora haja uma heurística para ela (*Loop Body Execution Time Relation*).