Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 2, N. 1, 2014.
Trabalho apresentado no CMAC-Sul, Curitiba-PR, 2014.

Otimizações Computacionais Aplicadas a um Simulador de Ondas Aquáticas

Jean M. B. Oliveira, **Mauricio F. Gobbi**,
Depto de Engenharia Ambiental, UFPR.
81531-990, Campus Centro Politécnico, Curitiba-PR.
E-mail: jmichaelb86@hotmail.com, gobbi@ufpr.br

Resumo: *O programa FUNWAVE é amplamente utilizado no mundo da engenharia costeira para a simulação numérica de problemas relacionados a propagação de ondas aquáticas em regiões costeiras, através da implementação de modelos do tipo Boussinesq. Entretanto, a simulação precisa de fenômenos reais de larga escala pode se tornar computacionamente muito cara, exigindo uma grande quantidade de recursos computacionais e de tempo de processamento. No presente trabalho, técnicas de otimização computacional e paralelização são aplicadas a este programa com o objetivo de melhorar sua performance na execução de modelos de larga escala. O processo de otimização serial consiste na aplicação de técnicas básicas manuais de otimização, utilização de otimização automáticas provenientes do compilador IFortran e utilização da biblioteca de alto desempenho MKL. A paralelização é aplicada através da utilização da biblioteca padrão OpenMP, responsável pela paralelização de aplicações em sistemas de memória compartilhada. Os resultados das otimizações são comparados através da execução de exemplos clássicos de simulação de ondas em diferentes instâncias do programa. A máquina utilizada para as simulações foi o cluster SGI® ALTIX® XE 1300, propriedade da Universidade Federal do Paraná. Através das otimizações efetuadas neste trabalho, o tempo de execução médio do programa é reduzido para um sexto do tempo de execução inicial.*

Palavras-chave: otimização computacional, FUNWAVE, modelo de Boussinesq, OpenMP.

1 Introdução

Modelos do tipo Boussinesq têm sido propostos por décadas para simular a propagação de ondas em diversos regimes, que abrangem desde águas profundas até águas rasas. Estes modelos fornecem, dentro das suas regiões de validade, resultados que simulam com boa acurácia uma série de fenômenos importantes na engenharia costeira, desde ondas curtas de superfície geradas pelo vento a ondas longas como os tsunamis.

Apesar disso, a simulação de fenômenos reais em grandes regiões se torna computacionalmente muito cara, exigindo uma grande quantidade de memória e tempo computacional dos computadores utilizados, o que restringe o uso operacional em tempo real dos modelos.

Técnicas de otimização computacional têm como principal objetivo a otimização e paralelização de aplicações (programas) computacionalmente intensivas, sempre visando a melhora de sua performance. Dessa forma, o conhecido programa FUNWAVE, o qual é uma implementação computacional de modelos tipo Boussinesq escrita na linguagem Fortran, é um ótimo candidato a aplicação de tais técnicas, devido a sua grande complexidade e importância no mundo da engenharia costeira aplicada.

A primeira versão do programa foi escrita a mais de dez anos atrás, e possuia o único objetivo de implementar de forma correta os modelos tipo Boussinesq existentes, sem a preocupação com questões referentes ao desempenho e otimização de performance. As versões seguintes, apesar de terem incorporado os avanços teóricos subsequentes do modelos, mantiveram a mesma base de programação e uma estrutura semelhante à da versão original.

DOI: 10.5540/03.2014.002.01.0019

Dessa forma, o objetivo principal do presente trabalho é a otimização da performance e a subsquente paralelização do programa FUNWAVE, através da utilização de técnicas conhecidas de Computação de Alto Desempenho e do uso da biblioteca padrão OpenMP para paralelização em sistemas de memória compartilhada. A aplicação de tais técnicas é feita, no entanto, sem a intenção de se alterar drasticamente a estrutura principal do programa, já bem conhecida e aceita no mundo da engenharia costeira aplicada.

2 O programa Funwave

A primeira versão do programa, FUNWAVE 1.0, foi desenvolvida em 1998 pelo Centro de Pesquisa Costeira Aplicada, Universidade de Delaware, com o objetivo de se poder simular computacionalmente fenômenos relativos à propagação de ondas aquáticas de superfície em regiões costeiras [8]. As equações de Boussinesq altamente não-lineares e o modelo numérico de ordem mais alta de Wei *et al.* [13] foram utilizados como base para o programa.

Após o lançamento do FUNWAVE 1.0, trabalhos adicionais de extensão e melhoras dos modelos de Boussinesq anteriores continuaram sendo publicados, tanto na parte teórica, como na parte numérica e computacional. Todas essas melhoras e diferenças na formulação foram incorporadas na versão 2.0 do programa [9] e esta é a versão que está sendo utilizada no presente trabalho. A lista das melhoras fornecidas pela versão 2.0 em relação à versão inclui: sistema de coordenadas curvilíneas generalizadas; múltiplos níveis móveis de referência; sistema de grid escalonado; gerador de onda em uma direção; fronteiras internas; termo de vorticidade vertical mantido nas equações governantes.

Além desses incrementos, as equações desta versão foram implementadas de uma forma generalizada que permite ao usuário escolher (através de parâmetros de entrada do programa) entre diferentes modelos de Boussinesq para a simulação de um problema. Dessa forma, o usuário pode escolher o modelo mais apropriado para a simulação de um problema específico e pode também comparar a eficiência e acurácia de cada modelo. Dentre os modelos disponíveis, estão os modelos linear e não-linear de águas rasas, o modelo de Boussinesq padrão e os modelos de Nwogu [10], Wei [13], Kennedy [7], Chen [4] e Gobbi [6].

O programa FUNWAVE é bastante complexo e possui mais de 15400 linhas de código e é constituído por uma rotina principal e mais de 140 subrotinas responsáveis pelas mais diversas tarefas: leitura e inicialização dos dados, cálculo das estruturas do domínio, geração de ondas no interior do domínio, cálculos dos termos das equações, cálculos das derivadas dos termos utilizando o método das diferenças finitas, montagem e solução iterativa dos sistemas de equações, checagem de erro, impressão de dados, geração de arquivos de saída, entre outras. Além do mais, uma grande quantidade de parâmetros e constantes devem ser ajustados a cada nova simulação, sendo 10 arquivos de entrada necesários à execução do programa e 9 arquivos de saída gerados. Os arquivos de entrada especificam parâmetros intrínsecos do programa, batimetria do domínio, mapeamento de coordenadas, tipo físico de cada elemento da grade, elevação inicial da superfície livre da água, velocidades horizontais iniciais, espectros de frequência e direção das ondas, camadas esponja (responsáveis pela dissipação de energia), entre outros dados necessários. Os arquivos de saída do programa são 9 e armazenam dados referentes às velocidades espaciais, elevação da superfície, pressão e vorticidade calculadas. As grandezas de saída são calculadas tanto em medidores espaciais, distribuídos ao longo do domínio, quanto em medidores temporais fixados em intervalos de tempos específicos. Além dos valores das grandezas que caracterizam o movimento de ondas simulado, imagens ilustrativas referentes à propagação das ondas também são geradas nos arquivos de saída.

Como pode ser observado em [8, 3, 11, 1], o programa obteve bons resultados nas simulações de diversos fenômenos relacionados ao movimento de ondas se propagando em profundidades pequenas e intermediárias em regiões costeiras, resultados esses que foram comparados tanto com dados experimentais como com previsões teóricas. Dentre os principais fenômenos simulados, pode-se citar: quebra, empinamento e efeito de *runup* de ondas; interações não-lineares entre

ondas e entre correntes e ondas; dispersões de frequência e amplitude; difracção e refracção; dissipação de energia como consequência da rebentação e da geração de harmônicas. Todas essas características fazem com que o programa FUNWAVE possua um razoável "peso"computacional, seu tempo de execução pode variar de alguns segundos a dias de processamento, dependendo da complexidade do problema que se está simulando. Todas essas questões justificam a intenção desse trabalho de melhorar a performance do programa.

3 Otimização e Paralelização

As principais técnicas de otimização computacional são: otimizações automáticas efetuadas pelos compiladores, utilização de *softwares* especializados em avaliar diferentes aspectos da performance de aplicações (*profilers*), algoritmos computacionais mais eficientes, utilização de bibliotecas de alto desempenho, entre outras. A paralelização deve ser encarada como um último passo no processo de otimização, pois deve ser aplicada apenas após o programa estar rodando serialmente da maneira mais otimizada possível.

O método utilizado aqui consistiu na utilização de um *profiler* como auxiliar em todo o processo de otimização, o *software* Vtune. Esse *software* é fornecido pela Intel® e, portanto, as informações fornecidas por ele são especializadas para os processadores desse fabricante, entretanto, processadores de outras marcas também são permitidos. Os dados fornecidos pelo Vtune foram fundamentais para esse trabalho, permitindo avaliações do desempenho inicial do programa, avaliações parciais durante o processo de otimização e comparações finais entre a performance do programa inicial e da sua versão otimizada. Através da utilização do Vtune, pôde-se obter um perfil geral da performance inicial do programa e identificar as subrotinas e regiões que consumiam uma porcentagem maior de recursos computacionais e do tempo de execução total do programa (*hotspots*). Com base nessas informações, o plano de execução desse trabalho consistiu então nas seguintes etapas:

- aplicação de otimizações manuais nas porções críticas do código;
- utilização de otimizações automáticas do compilador IFortran (Intel® Fortran);
- utilização da biblioteca de alto desempenho MKL para a substituição de funções de custo computacional elevado;
- paralelização do programa através da biblioteca OpenMP;

As otimizações manuais aplicadas ao programa FUNWAVE foram:

Strength reduction Essa técnica consiste em se explorar o código buscando expressões que possuem um custo computacional elevado e substituí-las por alternativas mais baratas. Expressões como `x**2.0` não são otimizadas e o seu resultado implica na avaliação de uma exponencial e um logaritmo. Uma alternativa simples é a substituição por `x*x`, resultando em uma economia de muitos ciclos de *clock*.

Eliminação de subexpressões Nessa técnica, o que se faz basicamente é pré calcular partes invariantes de expressões contidas em *loops*, ou que se repetem ao longo do código, e atribuir os valores desse cálculo a variáveis temporárias.

Substituição de variáveis indexadas Variáveis indexadas (*arrays*) se valem do **princípio da localidade** que, resumidamente, diz que quando um elemento do de um *array* é carregado para a memória *cache*, os elementos vizinhos também o são, saturando essa memória de pequena capacidade de armazenamento com dados muitas vezes desnecessários. Assim, o uso contínuo de *arrays* pode aumentar consideravelmente o número de erros de *cache* (*cache misses*) implicando em um uso ineficiente da memória e diminuindo a performance de aplicações. Dessa forma, sempre quando possível, essas variáveis devem ser substituídas por variáveis escalares.

DOI: 10.5540/03.2014.002.01.0019

Inlining Essa técnica consiste em substituir a chamada de uma subrotina pelo corpo de código que a compõe, eliminando o *overhead* (custo) da chamada. Chamadas constantes a rotinas que utilizam e modificam variáveis de tamanho grande podem implicar em latências associadas ao carregamento dessas variáveis da memória e ao armazenamento temporário, via uso de registradores, de seus valores modificados pelas subrotinas, além de criarem desconexões entre blocos adjacentes de código, inibindo otimizações que poderiam ser efetuadas pelo compilador, bem como pelo próprio programador.

Otimização de *loops* Na maioria das aplicações numericamente intensivas, *loops* são onde a maior porcentagem do tempo de execução é gasto. Se a estrutura do código dentro dos *loops* for simples, o compilador pode ser hábil o suficiente para gerar versões mais rápidas destes. Do contrário, a complexidade do código pode impedir a atuação do compilador e, assim, deve-se trabalhar manualmente para que eles possam ser otimizados. As técnicas de otimização de *loops* utilizadas foram *loop unrolling* (desenrolar de laços), inversão de *loops* e eliminação de desvios condicionais (IFs) de dentro dos *loops*.

Como foi comentado, as otimizações manuais são aplicadas diretamente no código e, por esse motivo, exigem um razoável esforço e conhecimento da linguagem e da aplicação em questão por parte do programador. A maioria das técnicas manuais utilizadas são efetuadas automaticamente pelos compiladores atuais, entretanto, a complexidade do código e forma como este está estruturado podem inibir a aplicação automática dessas otimizações. Além do mais, o compilador não pode avaliar a efetividade das técnicas, podendo aplicá-las mesmo que haja uma perda de performance. Portanto, a abordagem mais aconselhável é a aplicação tanto de técnicas manuais, como a utilização auxiliar das automatizações oferecidas pelos compiladores. Essa abordagem foi utilizada no presente trabalho e algumas técnicas, como *loop unrolling* e outras otimizações de *loops* foram aplicadas com a ajuda do compilador IFortran.

Além das otimizações, também se fez o uso da biblioteca de alto desempenho MKL (*Math Kernel Library*). Essa biblioteca fornece uma grande quantidade de rotinas e funções otimizadas e já paralelizadas que executam diversas operações matemáticas envolvendo vetores e matrizes. Isso é feito através de chamadas à outras bibliotecas, como a BLAS (*Basic Linear Algebra Subprograms*), *Sparce* BLAS, LAPACK (*Linear Algebra Package*) e ScaLAPACK, PBLAS (*Parallel* BLAS) e VML (*Vector Math Library*). Além disso, ela também inclui funções estatísticas, transformadas de Fourier, ferramentas para equações diferenciais parcias, entre outras facilidades. As rotinas fornecidas são especializadas para obter alta performance em processadores Intel®. Nesse trabalho, utilizou-ase principalmente as funções trigonométricas e logarítmicas provenientes da biblioteca VML, já que as versões dessas funções fornecidas pelo compilador IFortran possuem custo elevado e, assim, o uso das suas alternativas otimizadas resultou em uma melhora na performance geral do programa.

A etapa final do processo de otimização do programa FUNWAVE consiste na sua paralelização utilizando a biblioteca OpenMP. A OpenMP (*Open Multi-Processing*, ou multiprocessamento aberto) é uma API (*Application Programming Interface*) representando uma coleção de diretivas, bibliotecas de rotinas e de variáveis de ambiente destinadas à programação paralela baseada em *threads* em sistemas de memória compartilhada. O processo de paralelização do programa ainda encontra-se em andamento, pois a paralelização baseada em *threads* é feita de forma incremental, rotina a rotina, tendo como prioridades iniciais as rotinas mais críticas e depois passando para as rotinas de menor custo computacional.

Na proxíma seção os resultados da otimização serial e os resultados parciais da paralelização são apresentados.

4 Resultados

As diferentes instâncias do programa FUNWAVE foram executadas no cluster SGI® ALTIX® XE 1300, propriedade da Universidade Federal do Paraná (UFPR). Para a programação para-

lela, foi utilizado um nó ALTIX® XE 320, contendo dois processadores Intel® Xeon® E5462 Quadcore, ou seja, 8 *threads* foram utilizados para executar o programa em paralelo.

Foram executados três exemplos diferentes pelo programa FUNWAVE: Berkhoff [2], *Waves in a Circular Channel* [5] e *Waves Passing a Detached Breakwater* [12] para comparação dos resultados. A Tabela 1 fornece os resultados apenas das otimizações seriais (sem a paralelização), fornecidos pelo programa Vtune. O objetivo dessa primeira tabela é a comparação entre os números de ciclos de *clock*, de instruções, das taxas CPI (razão entre o número de ciclos de *clocks* e o número de instruções) e de erros de *cache* das versões otimizadas e iniciais de cada exemplo. Todas elas constituem medidas padrão para avaliações de performance de aplicações. Todos os exemplos apresentaram uma diminuição média de aproximadamente 45% do número de ciclos de *clock*, passaram a executar aproximadamente duas instruções a cada ciclo, fazendo um uso melhor do processador. Além do mais, o programa passou a utilizar de forma mais eficiente a memória, isso pode ser confirmado nas taxas de erros de *cache* de cada exemplo, as quais foram reduzidas para menos da metade em cada um dos casos, em comparação à versão inicial.

Tabela 1: Exemplos FUNWAVE - Desempenho original e otimizado

	Ciclos de clock	Nº de instruções	CPI	L1 cache miss rate
Detach_original	4.349.232.416.000	5.581.135.408.000	0,779	0,014
Detach_otimizado	2.317.628.032.000	4.399.639.184.000	0,527	0,006
Circular_original	3.906.583.152.000	5.216.617.472.000	0,749	0,011
Circular_otimizado	2.255.718.224.000	4.105.150.520.000	0,549	0,005
Berkhoff_original	3.710.897.456.000	5.283.441.200.000	0,702	0,012
Berkhoff_otimizado	2.285.324.592.000	4.270.486.848.000	0,536	0,005

Além dos resultados da otimização manual, a Tabela 2 fornece também os resultados da paralelização através do uso da biblioteca OpenMP.

Tabela 2: Tempos da execução

Exemplo	T_0	T_s	T_p	T_0/T_s	T_s/T_p	T_0/T_p
Detach	133m26.832s	52m38.652s	18m5.875s	2,5	2,9	7,4
Circular	78m3.564s	43m20.253s	13m57.001s	1,8	3,1	5,6
Berkhoff	368m21.816s	191m11.198s	62m45.408s	1,9	3,0	5,9

Onde T_0, T_s, e T_p representam os tempos das versões original, otimizada serial e otimizada paralela, respectivamente. As relações entre os tempos de execução de cada versão são fornecidas a partir da quarta coluna. Os *speedups*, representados pela relação T_s/T_p, mostram que as versões paralelas executaram aproximadamente três vezes mais rápidas que as versões seriais. O exemplo *Waves Passing a Detached Breakwater* foi o que mais se beneficiou das otimizações seriais e também o que obteve o maior ganho do tempo paralelo em relação à versão original, a versão otimizada paralela desse exemplo se mostrou ser mais de 7 vezes mais rápida que a versão original. As outras versões finais se mostraram ser aproximadamente 6 vezes mais rápidas que suas versões originais correspondentes.

Através da análise dos resultados, pode-se concluir que o objetivo inicial da otimização serial do programa foi concluído, pois o programa passou a executar aproximadamente duas instruções por ciclo, o que é considerado bom para um programa do porte do FUNWAVE. Além do mais, apenas com essas otimizações, o tempo de execução caiu para aproximadamente metade do tempo inicial. Como foi comentado, a paralelização do programa encontra-se em andamento e, como 8 processadores estão sendo utilizados para execução paralela, uma diminuição de um terço do tempo da versão otimizada em relação à versão otimizada paralela ainda está um pouco

distante do *speedup* ideal que deveria ser atingido.

Finalmente, levando-se em conta a complexidade e a intensividade computacional do programa FUNWAVE, no qual já se observou modelos complexos que demoraram mais de uma semana completa para serem executados, uma redução média total de 6 vezes no tempo de execução pode ser considerada boa.

Referências

[1] Barreto, F. T. C. "Aplicação do modelo de ondas FUNWAVE usando espectro direcional de onda para determinar a distribuição espacial de ondas irregulares em Praia Mole-ES". Trabalho de Conclusão de Curso. UFES, 2011.

[2] Berkhoff, J.C.W., Booy, N. e Radder, A.C. Verification of numerical wave propagation models for simple harmonic linear water waves. *Coastal Engineering*, 1982. 6, 255-279.

[3] Bruno, D., Serio, F. e Mossa, M. The FUNWAVE model application and its validation using laboratory data. *Coastal Engineering*, 2009. 56(7), 773-787.

[4] Chen, Q., Kirby, J.T., Dalrymple, R.A., Kennedy, A.B. e Chawla, A. Boussinesq modeling of water transformation, breaking and runup. II: Two horizontal dimensions. *Journal of Waterway, Port, Coastal and Engineering*, 2000. 126, 48-56.

[5] Dalrymple, R.A. e Kirby, J.T. Spectral methods for forward-propagation water waves in conformally-mapped channels. *Applied Ocean Research*, 1994. 16, 249-266

[6] Gobbi, M.F., Kirby, J.T., Wei, G. A fully nonlinear Boussinesq model for surface waves. Part 2. Extension to $O(kh)^4$. *Journal of Fluid Mechanics*, 2000. 405, 182-210.

[7] Kennedy, A.B., Chen, Q., Kirby, J.T., Dalrymple, R.A. Boussinesq modeling of water transformation, breaking and runup. I: One dimension. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 2000. 126, 39-47.

[8] Kirby, J.T., Wei, G., Chen, Q., Kennedy, A.B., Dalrymple, R.A. "FUNWAVE 1.0 Fully nonlinear Boussinesq wave model. Documentation and user's manual". Research Report CACR-98-06. Center for Applied Coastal Research, 1998. University of Delaware.

[9] Kirby, J.T., Wei, G., Shi, F. "FUNWAVE 2.0 Fully nonlinear Boussinesq wave model on curvilinear coordinates. Documentation and user's manual". Research Report CACR-03-xx. Center for Applied Coastal Research, 2005. University of Delaware.

[10] Nwogu, O. An alternative form of the Boussinesq equations for nearshore wave propagation. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 1993. 119, 618-638.

[11] Palha, A., Fortes, C. J. e Mase, H. Numerical simulation of random wave runup on seawall near shoreline with FUNWAVE model. *Proceedings of the Seventeenth International Offshore and Polar Engineering Conference*. Lisboa-Portugal, 2007.

[12] Watanabe, A. e Maruyama, K. Numerical modeling of nearshore wave field under combined refraction, diffraction and breaking. *Coastal Engineering in Japan*, 1986. 29.

[13] Wei, G., Kirby, J.T., Grilli, S.T. e Subramanya, R. A fully nonlinear Boussinesq model for surface waves. Part 1. Highly nonlinear unsteady waves. *Journal of Fluid Mechanics*, 1995. 294, 71-92.

