Portabilidade na Programação para o Processamento de Alto Desempenho*

Evandro Clivatti Dall'Agnol† Lucas Correia Villa Real‡
Epifanio Dinis Benitez§ Gerson Geraldo H. Cavalheiro

Programa de Pós-Graduação em Computação Aplicada
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
São Leopoldo – RS – Brasil
{ecd, lucasvr, epifanio, gersonc}@exatas.unisinos.br

Resumo

Este artigo apresenta estudos de caso avaliando Anahy, um ambiente de desenvolvimento e execução de aplicações paralelas em aglomerados. Este ambiente foi projetado para oferecer recursos para a exploração do processamento de alto desempenho através de uma ferramenta de programação capaz de retirar do programador a responsabilidade de gerenciar os recursos disponíveis na arquitetura. A estrutura básica de Anahy é apresentada, bem como as ferramentas selecionadas para implementá-lo. Por fim, é apresentada uma análise de resultados obtidos com aplicações Anahy. A questão da portabilidade conduz as discussões neste trabalho. São considerados dois aspectos: a portabilidade de desempenho e a portabilidade de código.

1 Introdução

A popularização de aglomerados de computadores como suporte de hardware à programação paralela motivou um grande esforço de programadores na implementação de aplicações com alto custo computacional para esta arquitetura. Muitas destas aplicações foram desenvolvidas com ferramentas de programação oferecendo primitivas básicas para a exploração dos recursos computacionais dos aglomerados, como chamada de procedimentos remotos (RPC), *threads* e comunicação entre processos. Estas ferramentas, ditas *ferramentas clássicas* de programação concorrente [9] no contexto deste trabalho, possuem como grande vantagem o fato de serem facilmente encontradas nas diferentes configurações de aglomerados.

No entanto, nos agregados, encontram-se os principais problemas relacionados à portabilidade de desempenho: agregados são constituídos de nodos com capacidades distintas de processamento, ou seja, podem possuir diferentes características de hardware. Pelo fato destas ferramentas clássicas permitirem a exploração de forma especializada de um determinado recurso de processamento, a possibilidade de migração destas aplicações de uma arquitetura para outra é baixa. Uma alternativa é a dissociação entre a descrição da concorrência da aplicação e o paralelismo real disponível na arquitetura.

Neste artigo estes dois critérios de portabilidade (ferramentas e desempenho) são discutidos no contexto da concepção de um ambiente de processamento de alto desempenho. Anahy é um ambiente de concebido para prover ao usuário (programador) tanto uma interface de programação concorrente de alto nível como um núcleo executívo. As estratégias adotadas para sua implementação fazem uso da interface de programação POSIX para *threads* e de um núcleo de escalonamento.

O restante deste artigo está estruturado como segue. A Seção 2 discute abordagens encontradas na bibliografia para ambientes de processamento de alto desempenho. A Seção 3 apresenta as ferramentas selecionadas para implementação de Anahy segundo os critérios de portabilidade. A Seção 4 aborda o algoritmo de escalonamento adotado para ser implementado no núcleo executivo. Aplicações implementadas sobre Anahy e resultados de desempenho preliminares obtidos, sobre o protótipo em uma arquitetura SMP, são apresentados na Seção 5.

2 Portabilidade em Ambientes Paralelos

Atualmente, as questões do mapeamento das tarefas nos processadores, dos dados nos módulos de memória e do controle de execução, todas importantes para assegurar por-

*Projeto Anahy – CNPq (55.2196/02-9)
†ITI/CNPq
‡ITI/CNPq
§BIC/FAPERGS

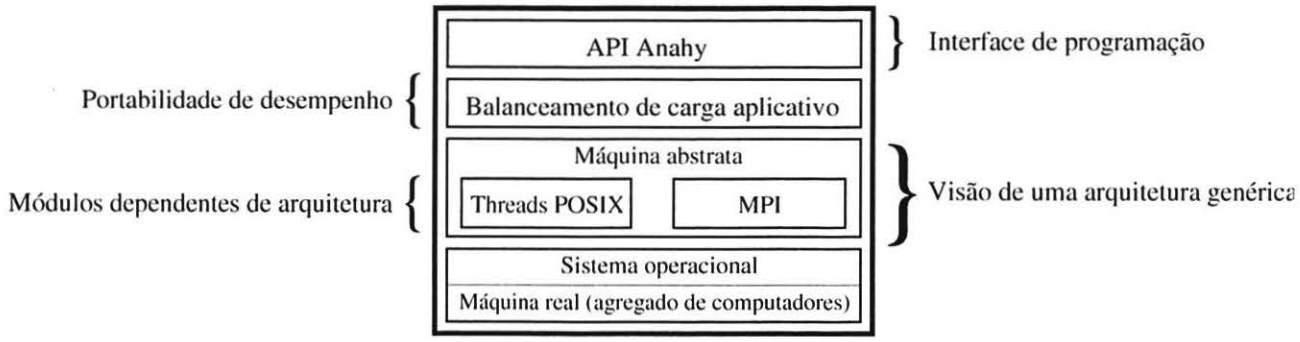

Figura 1. Relacionamento do escalonamento com a aplicação

tabilidade em ambientes paralelos, não são abordadas pela maioria dos ambientes de programação paralela existentes, como Athapascan-0 [3], PM^2 [10] e DPC++ [6]. Outros ambientes ainda, como Cilk [1], Jade [16] e Athapascan-1 [12], dispõem de recursos de programação através dos quais o programador é capaz de descrever a concorrência de sua aplicação e as trocas de dados com uma boa abstração do hardware disponível, porém todas elas apresentaram problemas de desempenho ([7, 15]).

Anahy, um ambiente de execução paralelo sobre aglomerados, propõe tanto uma interface de programação quanto um núcleo executivo capazes de controlar a execução das tarefas de forma a garantir uma semântica de execução coerente com a definida pelo programador. Um diferencial de *Anahy* é a preocupação com a questão do desempenho, objeto deste trabalho, permitindo que um programa desenvolvido usando esta esta interface aplicativa possa ser executado em diferentes configurações de aglomerados, usufruindo ao máximo dos recursos disponíveis.

O enfoque dado para o modelo da implementação do ambiente Anahy tem como premissa básica a dissociação da descrição da concorrência da aplicação do paralelismo disponível na arquitetura. Assim, o programador pode definir o número de atividades concorrentes de sua aplicação, não considerando os recursos de processamento disponíveis. A exploração da arquitetura é realizada por um núcleo executivo responsável pelo escalonamento. Este núcleo tem por função adaptar o número de atividades realizadas em paralelo de acordo com a capacidade de processamento disponível.

Para obter tal modelo, foi definida uma abstração de tarefa concorrente em nível usuário. Nesta abstração, uma *thread* consiste em um fluxo de execução não bloqueante que recebe parâmetros de entrada e que retorna resultados de seu processamento. Uma *thread* em Anahy, portanto, não realiza nenhuma operação de sincronização, exceto criação de novas *threads* e obtenção de resultados de retorno (tal como um *join*) de outras *threads*.

3 Portabilidade em Anahy

A estrutura de Anahy reflete as decisões de projeto as quais privilegiam as soluções dadas às questões de portabilidade. Esta estrutura pode ser visualizada na Figura 1, que destaca as diferentes camadas propostas para viabilizar a portabilidade de execução de uma aplicação concorrente sobre uma arquitetura do tipo aglomerado de multi-processadores. Estas camadas são discutidas a seguir.

A interface de programação (API) de Anahy oferece recursos para a implementação de programas concorrentes. O modelo proposto foi analisado segundo diferentes critérios considerados *úteis* para linguagens de programação concorrentes. Entre estes critérios está a minimização da dificuldade do programador em gerenciar um grande número de fluxos de execução concorrente e as comunicações entre estes fluxos.

Para permitir a portabilidade de desempenho, o núcleo executivo foi modelado [5, 8] de forma a suportar a implantação de diferentes algoritmos de escalonamento [4] na execução paralela e distribuída de programas. A idéia é adicionar ao ambiente a possibilidade de introduzir técnicas de balanceamento de carga através da adaptação do núcleo executivo para responder de forma adequada a diferentes critérios de regulação de carga (tempo de execução, consumo de memória, etc.) conforme as características da aplicação e da arquitetura.

Em Anahy, a questão clássica da portabilidade de código também foi considerada. Para a implementação dos módulos dependentes de arquitetura optou-se pelo uso de ferramentas de programação que possam ser facilmente encontradas nas mais diferentes configurações de aglomerados: *threads* POSIX e MPI/*sockets*. Essas ferramentas foram selecionadas por possibilitarem a exploração dos dois níveis de paralelismo de um aglomerado: intra e entre-nodos. Nota-se que este aspecto não envolve somente a linguagem de programação, mas também bibliotecas que permitam manipular efetivamente os recursos de uma arquitetura, tal como

um aglomerado.

Trabalhos de implementação foram realizados com o intuito de validar o modelo de programação proposto. As implementações foram realizadas utilizando ferramentas clássicas de programação (i.e., *threads* POSIX e MPI) restringindo o uso das potencialidades destas aos recursos necessários à implementação de programas Anahy. Alguns destes trabalhos podem ser encontrados em [2, 11] e em [14] encontra-se a descrição de uma aplicação real desenvolvida com o intuito de validar o conjunto do modelo.

4 Escalonamento em Anahy

Em um ambiente de execução paralela, é possível verificar que o escalonamento é realizado em dois níveis distintos: sistema e aplicativo. Enquanto o escalonamento no nível de sistema busca realizar a ocupação dos recursos computacionais de uma arquitetura para realizar a execução de um programa, o escalonamento no nível de aplicativo tem como preocupação a distribuição da carga computacional gerada pelo programa sobre os diferentes recursos de execução da arquitetura. Anahy é um ambiente de execução que desenvolve o escalonamento aplicativo (*cf.* [7], cap. 2).

Um algoritmo de escalonamento deve ser capaz de responder a questões simples que dizem respeito à alocação de tarefas aos processadores (e eventualmente à migração de tarefas) para realizar sua função básica: executar o programa em um tempo finito. Em Anahy, a solução destas questões são buscadas por políticas que explorem as características do programa a ser executado e da arquitetura destino sobre a qual a execução é realizada. Desta forma, um algoritmo de escalonamento é composto por dois subsistemas internos: um de *manipulação de informação*, responsável por observar a carga do sistema, oferecendo subsídios para um segundo sub-sistema, o de *decisão*, responsável por manipular as tarefas do programa e sua execução sobre a máquina.

A solução para garantir que Anahy seja um ambiente de execução que assegure a portabilidade de desempenho dos programas em execução passa pela definição de um módulo de escalonamento que seja independente da aplicação. Desta forma, alterar o algoritmo de escalonamento, adequando a política de tratamento das tarefas às características da aplicação e da máquina destino, é uma operação que pode ser feita sem a necessidade de modificar o código do programa submetido. Na estratégia adotada, o núcleo executivo limita o número de atividades concorrentes da aplicação em execução simultânea. Isto significa que, em um dado momento, o número máximo de atividades em execução é limitado em função dos recursos computacionais disponíveis. Este limite é dado pelo número de *processadores virtuais*, ou PVs, ativos no núcleo. Assim, o programador pode definir um número de atividades concorrentes que ultrapasse as capacidades da arquitetura disponível: cabe ao núcleo executivo ativar a execução destas atividades sobre os recursos de processamento virtuais disponíveis.

O escalonamento em Anahy é realizado através da manipulação de um grafo de fluxo de dados, exemplificado na Figura 2. Este grafo apresenta as dependências de dados entre as tarefas. No exemplo dado, uma *thread* principal (nível 0) cria outras três *threads* que formarão o nível 1 e essas, por sua vez, criam outras e assim sucessivamente. Esta representação possibilita a visualização de quais tarefas precisam ser concluídas para que seus dados sejam utilizados por outras, ou seja, as tarefas "filhas" serão executadas por completo para que depois as tarefas que as criaram possam utilizar os dados gerados e concluir sua execução. Na Figura 2, nota-se pelos tons de cinza quais tarefas estão sendo executadas, quais já foram executadas e quais estão bloqueadas esperando sua vez de executar. Nota-se que a ordem de execução é da direita para a esquerda e de baixo para cima, por que essa é a ordem da dependência de dados entre as tarefas.

A estratégia do escalonador de executar as tarefas nesta ordem resulta em uma exploração da localidade de tarefas. A exploração desta localidade tem por conseqüência uma diminuição no número de sincronizações necessárias para o controle do avanço da execução.

A utilização deste modelo mostrou-se eficiente para a execução de tarefas com alto grau de paralelismo. Algoritmos de escalonamento estão sendo desenvolvidos para manipular, de diferentes formas, esse grafo.

5 Resultados de Desempenho em Aplicações Anahy

Trabalhos de implementação foram realizados com o intuito de comparar o tempo de execução para três algoritmos fazendo uso de *threads* POSIX e do ambiente Anahy. Para isto, duas ferramentas foram escritas: um *Ray-Tracer*, onde um cenário descrito através de objetos geométricos é renderizado, e uma implementação paralela para compactação de arquivos baseada na *Zlib*. Uma terceira implementação teve por objetivo validar o uso de Anahy em uma situação em que a aplicação possui um elevado grau de concorrência e um grande número de sincronizações: o cálculo de um valor na série de Fibonacci.

Nestes testes, o hardware utilizado foi um Pentium IV de 1.8GHz com 512Mb de memória RAM, em relação à arquitetura mono-processada. A arquitetura bi-processada é composta de 2 processadores XEON (Quad Xeon *Hyper-threaded*) de 2.4GHz, operando com 2.5GB de memória RAM. Os testes foram compilados utilizando o GCC 3.2.2, rodando sobre o sistema operacional GNU/Linux, com kernel 2.4.20.

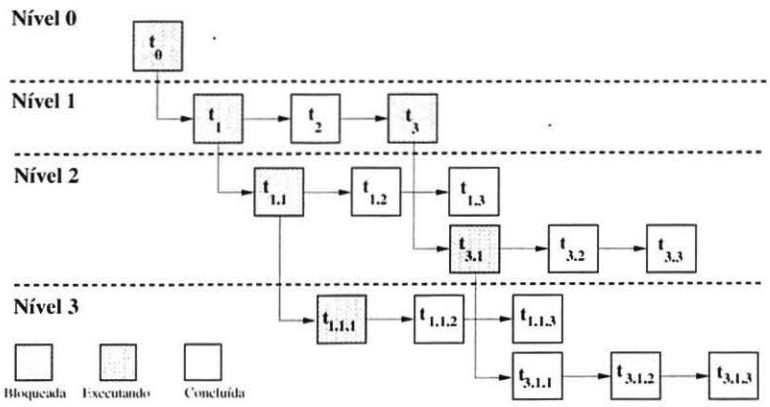

Figura 2. Grafo de dependência de dados

5.1 Ray-Tracer

A implementação do *Ray-Tracer* foi realizada de forma a dividir o cálculo da iluminação e determinação de cor de uma determinada região de uma cena entre diferentes *threads*. Assim, cada *thread* recebe uma quantidade determinada de linhas consecutivas a serem processadas e o seu processamento é retornado à uma *thread* responsável pela escrita no disco. O número de tarefas a serem criadas foi fixado em 256, utilizando um ambiente de descrição de cenário com resolução de $800x800$.

Deve-se notar que, nesta aplicação, a carga de processamento atribuída às *threads* é irregular. Ou seja, apesar de cada *thread* receber o mesmo número de linhas para serem calculadas, a carga computacional depende dos objetos contidos nesse conjunto de linhas: quanto mais objetos, mais cálculo é realizado na *thread*.

O esquema das *threads* criadas pela aplicação encontra-se na Figura 3. Como mostra a figura, não existe dependência entre as tarefas que não a sincronização presente ao final do processamento do conjunto de linhas.

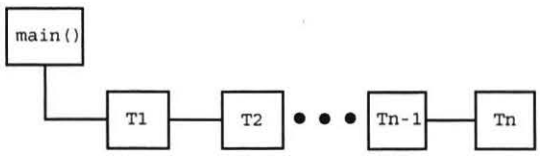

Figura 3. Modelo de tarefas independentes: Ray-Tracer e agzip

Os resultados obtidos na execução deste aplicativo encontram-se nas Tabelas 1, 2, 3 e 4. Nestas tabelas são apresentados os tempos, em segundos, e os desvios padrões obtidos para 100 execuções da aplicação em diferentes situações. As colunas PVs (processadores virtuais) indicam o grau de concorrência da aplicação suportado pelo núcleo executivo de Anahy.

Tabela 1. Tempos em segundos para a execução (seqüencial) do Ray-Tracer

Arquitetura	Média	Desvio Padrão
Mono-proc	131,615	0,126
Bi-proc	104,922	7,173

Tabela 2. Tempos em segundos do Ray-Tracer com PThreads

Arquitetura	Média	Desvio Padrão
Mono-proc	181,799	0,115
Bi-proc	50,646	0,460

Os resultados indicam que o ambiente Anahy é capaz de manipular com sucesso a quantidade de trabalho associada ao modelo de paralelismo do Ray-Tracer: em uma arquitetura mono-processada Anahy permitiu uma execução sem introduzir *overheads* em relação à execução seqüencial. Esta conclusão foi obtida comparando os resultados da Tabela 1 (execução seqüencial) com os das Tabelas 3 e 4 (execuções com Anahy). Já a execução da implementação utilizando PThreads, resultados apresentados na Tabela 2, introduz *overhead* significativo na execução em arquitetura mono-processada, possibilitando ganho de desempenho na execução na arquitetura bi-processada. No entanto, observa-se que Anahy permite obter um maior ganho de desempenho em relação à implementação com PThreads mesmos em arquiteturas bi-processadas devido ao fato de restringir a geração de custos adicionais na criação de *threads* gerenciadas pelo sistema.

Tabela 3. Tempos em segundos para a execução do Ray-Tracer em Anahy em uma arquitetura mono-processada.

PVs	Média	Desvio Padrão
1	131,552	0,124
2	131,542	0,118
3	131,550	0,127
4	131,543	0,122
5	131,533	0,120
10	144,066	0,105
15	138,328	0,116
20	138,504	0,122

Tabela 4. Tempos em segundos do Ray-Tracer em arquitetura bi-processada com Anahy

PVs	Média	Desvio Padrão
1	95,180	3,3016
2	55,229	4,5509
3	42,216	1,4201
4	36,781	0,5676
5	37,452	3,1633
10	35,760	0,3675
15	37,627	0,8435
20	28,923	0,7287

5.2 Compactação de arquivos

Outra aplicação realizada consiste na paralelização de um algoritmo de compactação de arquivos. Pelo fato de a compactação de dados ser uma operação bastante freqüente, esta é uma tarefa onde o ganho de desempenho faz-se desejável. A proposta de implementação foi de dividir o arquivo original em *streams* de mesmo tamanho entre as *threads* da aplicação, onde cada uma delas realiza o cálculo CRC-32 de seu bloco e realiza a compressão de sua *stream*. Para manter compatibilidade com aplicativos como GZip, o mesmo formato precisou ser utilizado, o que impôs a limitação de que a escrita em disco teria uma ordem pré-determinada. Assim, não é possível realizar a escrita em disco do resultado produzido por uma *thread* $N + 1$ caso a *thread* N ainda não tenha sido sincronizada, fazendo com que a escrita seja seqüencial. Este mesmo algoritmo foi utilizado com o uso de *threads* POSIX. O esquema de execução das tarefas desta aplicação é similar ao *Ray-Tracer*, o qual pode ser visto na Figura 3, sendo igualmente irregular.

Para a execução seqüencial, foi utilizada implementação já existente do GZip. Os resultados obtidos na execução desta aplicação encontram-se nas Tabelas 5, 6, 7, 8 e 9, considerando a compressão de um arquivo binário de 300MBs. Os tempos são apresentados em segundos, sendo

Tabela 5. Tempos em segundos da execução (seqüencial) do compactador GZip

Arquitetura	Média	Desvio Padrão
Mono-proc	43,698	2,829
Bi-proc	46,104	3,561

as médias e os desvios padrões obtidos de 100 execuções em cada situação. Os tempos de acessos a arquivos não foram considerados nas tomadas de tempos. Note-se que a implementação concorrente possui uma diferença em relação à versão seqüencial: nesta última é mantido um histórico da taxa de compressão de todo o arquivo, o que não é viável na implementação concorrente. Esta diferença resulta em uma maior complexidade (tempo de processamento) do algoritmo seqüencial. Os tempos da execução seqüencial encontram-se na Tabela 5.

Observando os resultados apresentados pela execução da aplicação com PThreads (Tabelas 6 e 8) é possível concluir que a obtenção de ganho de desempenho somente é possível em situações onde o mapeamento de atividades concorrentes da aplicação encontra-se adaptado à capacidade de processamento da arquitetura – índices satisfatórios de desempenho foram obtidos com 5 a 10 *threads* executando sobre a arquitetura bi-processada. Corrobora o fato de que mesmo possuindo um algoritmo de complexidade menor, os tempos de execução em uma arquitetura mono-processada (Tabela 6) foram superiores aos tempos da execução seqüencial (Tabela 5).

Nas Tabelas 7 e 9 encontram-se os resultados para a execução no ambiente Anahy. Nestas tabelas, uma nova coluna foi inserida, indicando o número de atividades concorrentes definidas pela aplicação – estas atividades são mapeadas nos PVs para serem executadas. Destacam-se os resultados obtidos para execução com 1 tarefa e 1 PV. Os tempos obtidos são inferiores aos tempos equivalentes nas execuções com PThreads.Estes resultados são conseqüência da implementação do núcleo executivo de Anahy: de fato não é criada nenhuma *thread*, não existindo *overheads* de execução. Quanto aos demais resultados, observa-se que as escolhas dos números de tarefas e de PVs utilizados influencia no desempenho final de forma diferente em cada arquitetura.

5.3 Fibonacci

Os últimos resultados apresentados nesta avaliação referem-se à execução de uma aplicação com elevado número de atividades concorrentes e sincronizações entre estas. Trata-se da implementação recursiva de um algoritmo de Fibonacci, que tem seu fluxo de execução ilustrado na Figura 4. Nota-se que para um valor pequeno (5), a quan-

Tabela 6. Tempos em segundos do compactador paralelo com Pthreads em uma arquitetura mono-processada

Threads	Média	Desvio Padrão
1	54,924	0,224
2	53,440	0,957
3	53,030	1,111
4	52,349	0,919
5	52,394	1,026
10	51,896	0,592
15	51,976	0,509
20	51,744	0,428

Tabela 7. Tempos em segundos do compactador paralelo em Anahy em uma arquitetura mono-processada

PVs	Tarefas	Média	Desvio Padrão
1	1	48,988	2,003
	2	49,822	1,086
	3	53,070	2,559
	4	57,387	1,759
	5	61,465	3,859
2	1	49,824	3,859
	2	52,584	2,700
	3	54,745	1,894
	4	56,715	1,795
	5	57,750	35,117
3	1	48,898	2,158
	2	49,384	1,121
	3	53,437	2,333
	4	60,477	1,580
	5	61,750	4,73
4	1	46,054	1,651
	2	48,778	2,504
	3	51,425	1,107
	4	59,707	1,949
	5	59,917	3,114
5	1	46,432	1,934
	2	49,658	2,592
	3	54,787	2,099
	4	61,752	4,208
	5	63,922	3,815

Tabela 8. Tempos em segundos do compactador paralelo com Pthreads em uma arquitetura bi-processada

Threads	Média	Desvio Padrão
1	53,043	2,592
2	43,023	2,023
3	31,348	2,023
4	22,592	2,097
5	20,592	2,097
10	20,716	0,238
15	21,561	0,171
20	21,985	0,381

Tabela 9. Tempos em segundos do compactador paralelo em Anahy em uma arquitetura bi-processada

PVs	Tarefas	Média	Desvio Padrão
1	1	37,596	2,493
	2	35,185	0,735
	3	34,411	0,590
	4	34,446	0,217
	5	34,314	0,235
2	1	37,218	3,101
	2	30,645	6,474
	3	28,763	7,236
	4	24,053	7,066
	5	30,284	4,491
3	1	37,696	1,916
	2	26,823	2,331
	3	22,428	1,504
	4	21,292	1,326
	5	21,322	1,206
4	1	36,858	0,171
	2	24,438	1,817
	3	22,366	0,726
	4	22,274	0,725
	5	22,202	0,557
5	1	35,910	0,505
	2	28,156	0,431
	3	19,731	0,564
	4	24,465	0,416
	5	20,950	0,129

Tabela 10. Tempos em segundos para Pthreads em Fibonacci

Arquitetura	Fibo	Média	Desvio Padrão
mono proc	15	1,221	0,0540
	16	1,391	0,0584
bi proc	15	1,095	0,1092
	16	1,414	0,1873

tidade de criações e sincronizações de tarefas realizadas é considerável.

A análise feita compara a execução do algoritmo paralelo suportado por PThreads (Tabela 10) e por Anahy (Tabelas 11 e 12).

Na implementação realizada, cada invocação recursiva da função Fibonacci gera a criação de uma nova tarefa concorrente. Como o suporte sobre PThreads gera uma nova *thread* para cada atividade, o cálculo fica limitado a valores baixos da série (o número de *threads* suportadas pelo sistema operacional é limitado). Já em Anahy, o número de atividades em execução simultânea é controlado pelo número de PVs do núcleo.

Também chama atenção nos resultados da execução em arquitetura bi-processada que, aumentando o número de atividades concorrentes em execução simultânea (maiores números de PVs e de tarefas), o *overhead* gerado pelas

Tabela 11. Tempos em segundos de Fibonacci em Anahy em uma arquitetura mono-processada

PVs	Fibo	Média	Desvio Padrão
1	15	0,186	0,0819
	16	0,509	0,2502
	17	1,482	0,7466
	18	5,170	0,0092
	19	13,877	0,0526
	20	36,285	0,0862
2	15	0,179	0,0794
	16	0,501	0,2427
	17	1,461	0,7424
	18	5,204	0,0301
	19	14,042	0,0434
	20	36,866	0,0889
3	15	0,059	0,0295
	16	0,098	0,0621
	17	0,177	0,2004
	18	0,302	0,2926
	19	0,374	0,0832
	20	0,778	0,0625
4	15	0,055	0,0338
	16	0,132	0,0883
	17	0,284	0,1924
	18	0,528	0,1710
	19	0,743	0,1606
	20	1,788	3,3887
5	15	0,092	0,0582
	16	0,177	0,1084
	17	0,391	0,3147
	18	0,834	0,6495
	19	0,797	0,1365
	20	1,315	0,3752

Tabela 12. Tempos em segundos de Fibonacci em Anahy em uma arquitetura bi-processada

PVs	Fibo	Média	Desvio Padrão
1	15	0,171	0,0213
	16	0,443	0,0321
	17	1,239	0,1064
	18	3,634	0,1989
	19	10,429	0,4364
	20	27,829	1,0544
2	15	0,134	0,0151
	16	0,285	0,0404
	17	0,613	0,0639
	18	1,452	0,1982
	19	3,837	0,5850
	20	10,219	0,9775
3	15	0,162	0,0223
	16	0,337	0,0422
	17	0,723	0,0813
	18	1,749	0,1622
	19	4,621	0,5051
	20	11,900	1,5037
4	15	0,198	0,0277
	16	0,431	0,0680
	17	0,962	0,1759
	18	2,383	0,5147
	19	6,114	1,4462
	20	16,115	3,8569
5	15	0,221	0,0096
	16	0,495	0,0486
	17	1,146	0,0723
	18	2,885	0,3529
	19	7,535	0,4952
	20	19,486	1,3612

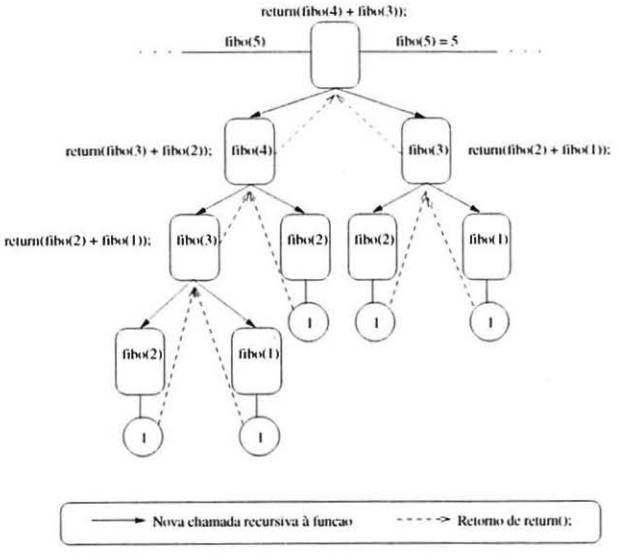

Figura 4. Modelo de execução de Fibonacci

sincronizações afeta o desempenho final. Este característica justifica, ainda mais, que o número de atividades concorrentes em execução simultânea deve ser adaptado aos recursos de processamento da arquitetura.

6 Conclusão

A portabilidade é uma das questões de grande importância no desenvolvimento de aplicações, abordada tanto em relação ao código como em relação ao desempenho. É desejável que a aplicação execute em diferentes arquiteturas e/ou sistemas operacionais e, além disso, execute eficientemente. Ou seja, que a aplicação seja capaz de utilizar eficientemente os recursos da arquitetura disponível.

Na programação para aglomerados de computadores, a portabilidade tem seu conceito estendido, tema abordado neste trabalho: um programa desenvolvido para uma arquitetura deste tipo deve poder ser executado independentemente da configuração que a arquitetura possa vir a assumir. Com a utilização de ferramentas tradicionais de programação concorrente ou paralela se torna difícil a obtenção deste nível de portabilidade, isso porque estas fer-

ramentas exploram recursos específicos de hardware. Para superar esta limitação, é necessário que a descrição da concorrência de uma aplicação seja independente das características do hardware a ser utilizado, como proposto por Anahy.

A portabilidade do próprio ambiente desenvolvido [13] também foi discutida neste artigo, em uma abordagem em relação às ferramentas adotadas na implementação de Anahy – ferramentas estas desenvolvidas em software livre e implementando recursos padronizados.

Atualmente, Anahy conta com um protótipo funcional oferecendo suporte à execução paralela sobre uma arquitetura SMP. A API fornecida permite a descrição da concorrência da aplicação através de primitivas de manipulação de *threads* (*athread_create* e *athread_join*). Os próximos passos abordarão o compartilhamento de dados e a repartição da carga gerada por um programa em execução em uma arquitetura com memória distribuída. Outros recursos clássicos de bibliotecas de *threads*, como mecanismos de sincronização (mutexes, por exemplo) serão incorporados à Anahy visando manter compatibilidade com códigos de aplicações já existentes.

Referências

[1] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In Shafi Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 356–368, Los Alamitos, CA, USA, November 1994. IEEE Computer Society Press.

[2] Adonize Bonetto e Fabio Mierlo. Modelagem concorrente para simulações monte carlo baseada no modelo de ising. In *Sessão de Pôsteres, ERAD 2002*, São Leopoldo, Brasil, Janeiro 2002.

[3] A. S. Carissimi. *Le noyau exécutif Athapascan-0 et l'exploitation de la multiprogrammation légère sur les grappes de stations multiprocesseurs*. Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, September 1999.

[4] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling general-purpose distributed computing systems. *IEEE Trans. on Software Engineering*, 14(2):141–154, February 1988.

[5] G. G. H. Cavalheiro, Y. Denneulin, and J.-L. Roch. A general modular specification for distributed schedulers. In LNCS 980 Springer Verlag, editor, *Proceedings of Europar'98*, Southampton, September 1998.

[6] G. G. H. Cavalheiro, R. C. Krug, S. J. Rigo, and P. O. A. Navaux. DPC++: An object–oriented distributed language. In *XV SCCC*, Arica, Chile, November 1995.

[7] Gerson G. H. Cavalheiro. *Athapascan-1: Interface générique pour l'ordonnancement dans un environnement d'exécution parallèle*. Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, September 1999.

[8] Gerson G. H. Cavalheiro. A general scheduling framework for parallel execution environments. In *Proceedings of SLAB'01*, Brisbane, Australia, May 2001.

[9] Gerson G. H. Cavalheiro. Introdução à programação paralela e distribuída. In Tiarajú A. Diverio e Philippe Navaux, editors, *I Escola Regional de Alto Desempenho*, Gramado, Janeiro 2001.

[10] Y. Denneulin. *Conception et ordonnancement des applications hautement irrégulières dans un contexte de parallélisme à grain fin*. Thèse de doctorat, LIFL – Université des Sciences et Technologies de Lille, Lille, France, October 1998.

[11] André Detsch, Guilherme B. Bedin, Hisham H. Muhammad, e Rafael G. Jeffman. Técnicas de treinamento concorrente de uma rede neural artificial multilayer perceptron. In *Sessão de Pôsteres, ERAD 2002*, São Leopoldo, Brasil, Janeiro 2002.

[12] F. Galilée, J.-L. Roch, G. G. H. Cavalheiro, and M. Doreille. Athapascan-1: On-line building data flow graph in a parallel language. In *Pact'98*, Paris, France, October 1998.

[13] Alex Sandro Garzão, Lucas Correia Villa Real, e Gerson G. H. Cavalheiro. Ferramentas para desenvolvimento de um ambiente de programação sobre agregados. In *Anais do Workshop em Software Livre*, Porto Alegre, Brasil, Maio 2001.

[14] Eduardo Moschetta, Fernando S. Osório, e Gerson G. H. Cavalheiro. Reconhecimento de imagens em aplicações críticas. In *III Workshop em Sistemas Computacionais de Alto Desempenho*, Vitória, Outubro 2002.

[15] K. H. Randall. *Cilk: Efficient Multithreaded Computing*. Thèse de doctorat, DEECS Massachusetts Institut of Technology, Massachusetts, June 1998.

[16] Martin C. Rinard, Daniel J. Scales, and Monica S. Lam. Jade: A high-level machine-independent language for parallel programming. *Computer*, 26(6):28–38, June 1993.