Algoritmos de Otimização para Simulação Distribuída de Arquiteturas Complexas

Joel G. Pereira, Rafael R. dos Santos, João Carlos Furtado
Departamento de Informática
Universidade de Santa Cruz do Sul (UNISC)
joelgp@gmail.com, {rsantos, jcarlosf}@unisc.br

Tatiana G. S. dos Santos
Centro de Excelência em Tecnologia
Eletrônica Avançada (CEITEC)
tatiana.santos@ceitec.org.br

Resumo

O projeto de arquiteturas de computadores complexas envolve diversas etapas e pode levar vários anos de desenvolvimento. Desse modo, é comum usar simulação com a finalidade de estudar os efeitos de novos mecanismos e das diversas modificações e atualizações na arquitetura, já que a implementação direta é proibitiva em função do tempo, custo e complexidade. Mesmo usando simulação, a busca pela melhor configuração pode ser onerosa em decorrência da diversidade de parâmetros que podem mudar e afetar o comportamento da arquitetura.

Este trabalho utiliza uma implementação de um algoritmo genético para a otimização e automatização da busca de configurações em simulação de arquiteturas de computadores. Os resultados mostram que, dentro do espaço de busca apresentado no artigo, o tempo de execução das simulações reduz em até 91%, se comparado à busca exaustiva. Além disso, os resultados encontrados pelo algoritmo genético representam valores com precisão acima de 97% com relação à solução ótima. Para diminuir ainda mais o tempo na obtenção dos resultados, o processamento das simulações executadas pelo algoritmo genético foi distribuído em um agregado e uma grade. Essa versão distribuída reduziu em mais de 80% o tempo de execução, quando comparado à execução seqüencial do mesmo algoritmo genético.

1 Introdução

O uso de simuladores para validar o desempenho de arquiteturas de microprocessadores complexos é essencial durante as fases preliminares de qualquer projeto, onde o balanceamento de recursos é um dos principais objetivos. Devido à complexidade das arquiteturas do estado-da-arte, obter resultados satisfatórios é um processo que envolve, além da execução das simulações, a análise dos resultados, que servem como base para novas configurações até que o objetivo seja atingido. Este processo envolve conseqüentemente inúmeras simulações através de um processo de refinamento sucessivo.

Tipicamente, a arquitetura necessita passar por um balanceamento a cada novo mecanismo ou mesmo no aprimoramento dos existentes, onde os recursos são re-adequados de acordo com as novas necessidades. Essa configuração deve ser definida com extremo cuidado pois pode significar o sucesso – ou falha – de uma nova arquitetura.

É comum que essas simulações sejam exaustivas ou ainda que compreendam a variação de apenas parte dos recursos envolvidos na arquitetura. Obter a melhor configuração possível através do processo exaustivo, ou seja, executando uma simulação para cada configuração de arquitetura possível, é impraticável pelo tempo envolvido na execução de todas as simulações e pelo volume de recursos computacionais geralmente envolvidos. A alternativa natural de limitar o espectro de variações, como a configuração da memória cache de instruções ou número de Unidades Funcionais (UFs), por exemplo, ainda não é a solução mais adequada já que ainda assim centenas de configurações são geradas e precisam ser validadas e comparadas. Além disso, não existe garantia alguma que a configuração ótima para a arquitetura seja alcançada.

Adicionalmente, cada configuração de arquitetura é simulada com vários *benchmarks*. Isso é de extrema importância dentro do contexto de microprocessadores de propósito geral, pois essas arquiteturas são avaliadas pelo desempenho médio de um conjunto de diferentes aplicações. Sem dúvida alguma, isso faz com que a quantidade de simulações necessárias aumente ainda mais.

No caso de um projeto de memória cache de instruções com dois níveis, por exemplo, onde cada nível é limitado a quatro parâmetros diferentes para capacidade e quatro parâmetros para associatividade, faz-se necessário simular em torno de quatrocentas configurações de arquitetura. Se um conjunto de cinco *benchmarks* é usado, o resultado somente pode ser obtido após duas mil simulações.

Desse modo, este trabalho propõe a utilização de um algoritmo genético (AG) para automatizar o processo de busca de configurações e diminuir o tempo de obtenção do resultado. Além disso, a execução do AG também foi distribuída ocasionando uma redução ainda maior do tempo de busca.

O presente trabalho está organizado da seguinte forma: na Seção 2 os trabalhos correlatos são apresentados. Na Seção 3 será apresentada a metodologia utilizada para otimizar a busca entre as diversas configurações de arquiteturas, enquando que a Seção 4 mostra detalhes da configuração na ferramenta desenvolvida. A Seção 5 mostra o ambiente de simulação, enquanto que a Seção 6 apresenta os resultados dessa pesquisa. Finalmente, na Seção 7, as conclusões e trabalhos futuros são mostrados.

2 Trabalhos Correlatos

2.1 Simulação de Arquiteturas Complexas

Como discutido anteriormente, a simulação de arquiteturas de microprocessadores é uma etapa indispensável no estudo, avaliação e definição de novos mecanismos para aumento de desempenho. Isso ocorre porque não é factível a implementação real de um microprocessador apenas para validar novas tecnologias. O projeto completo de um processador do estado-da-arte pode levar vários anos e não é possível refazê-lo a cada novo estudo.

Sendo assim, diversos simuladores já foram propostos com o intuito de promover a avaliação de uma determinada arquitetura. Uma das mais conhecidas e utilizadas ferramentas livres para esse fim, é a ferramenta SimpleScalar. Concebida inicialmente em 1997 [3] e implementada completamente em linguagem C, ela é amplamente aceita tanto na área acadêmica, quanto na indústria e já foi aprimorada diversas vezes, em vários estudos.

A ferramenta SimpleScalar ficou popular por ser simples de utilizar e bastante completa. Um dos seus simuladores, o *sim-outorder*, realiza a simulação completa ciclo-a-ciclo de arquiteturas superescalares. É possível configurar toda a arquitetura através de parâmetros passados ao simulador, tais como previsão de desvios, hierarquia de memória, unidades funcionais, etc. Por ser o simulador mais completo disponível no conjunto, é também o de menor desempenho devido ao seu nível de detalhamento da arquitetura e a quantidade de características implementadas [3]. Infelizmente, a simulação de uma única configuração pode levar até várias horas de processamento, devido à quantidade de instruções, complexidade da arquitetura e benchmark usado. Algumas propostas já foram estudadas com a finalidade de sanar, ou pelo menos, minimizar esse problema.

Uma alternativa de reduzir o tempo de execução de uma simulação é executar apenas parte dos *benchmarks* desejados. A ferramenta SimPoint busca pela parte mais representativa de um *benchmark* para uso no SimpleScalar. Isto é possível devido à característica dos programas apresentarem comportamentos que se repetem em um intervalo de tempo. Esta ferramenta une os segmentos do programa que apresentam comportamentos repetitivos, reduzindo o tempo de execução da simulação através da diminuição da quantidade de instruções a serem simuladas. A margem de erro, de acordo com testes realizados com os benchmarks da SPEC é menor do que 9% [8]. No entanto, para mecanismos avançados essa diferença pode ser crítica e não deve ser considerada como aceitável.

Neste trabalho o simulador *sim-outorder* é utilizado como referência por se tratar do simulador de código livre mais empregado neste tipo de experimentos. Adicionalmente, a ferramenta apresentada neste artigo, assim como o SimPoint, busca reduzir o tempo de execução de simulações com uma pequena margem de erro. A diferença é que o SimPoint reduz o tempo de execução através da otimização da aplicação executada, ao passo que neste trabalho o tempo de obtenção da arquitetura é reduzido de modo que atenda as necessidades do usuário, realizando várias simulações de forma otimizada pelo uso de um algoritmo genético.

Em 2001, Olivieri [11] fez um estudo sobre a aplicabilidade de AG no projeto de microprocessadores, usando como simulador o SimpleScalar. Foram analisados parâmetros referentes ao caminho de dados e a unidade de controle, com o espaço de busca superior a dez mil possibilidades. A solução ótima não foi obtida, possivelmente devido ao espaço de busca escolhido pelo autor. Este trabalho apresenta semelhanças com o artigo de Olivieri [11], como o uso de AG no projeto de microprocessadores e o uso do simulador SimpleScalar. Porém, o espaço de busca, benchmarks e cálculo do fitness são diferentes. Este trabalho também implementou a distribuição das simulações e buscou qualificar a configuração do microprocessador encontrada pelo AG, comparando-a com a configuração ideal.

2.2 Algoritmos Genéticos

O Algoritmo Genético, ou AG, inicialmente descrito em uma pesquisa desenvolvida por Holland em [10], é um algoritmo evolutivo baseado na teoria evolucionista de Darwin e objetiva melhorar uma população de indivíduos a medida que as iterações passam através de procedimentos como seleção, cruzamento e mutação. Este tipo de algoritmo é amplamente utilizado como heurística para casos em que a análise combinatória de todas as variáveis envolvidas em um dado problema é inviável pela alta demanda de tempo e custo computacionais [14].

Assim sendo, o presente trabalho utiliza este tipo de alternativa para evitar a busca exaustiva de configurações na definição do balanceamento de uma arquitetura qualquer. É

válido salientar que mesmo tratando-se de uma heurística, o AG atinge um nível muito próximo ao ótimo dentro de um espaço de tempo aceitável.

Além disso, o conceito aqui empregado baseia-se na idéia de hardware evolutivo, onde é possível aplicar sistemas evolucionários no auxílio de projetos de hardware. No presente trabalho, um algoritmo genético é utilizado para a evolução da configuração de um hardware. Através do uso de simuladores, não é necessário fazer protótipos do hardware para avaliar cada indivíduo da população do algoritmo genético. Neste tipo de aplicação, as configurações do hardware são geradas, simuladas, avaliadas, selecionadas automaticamente. Os melhores resultados continuam para as gerações seguintes, e no final, a melhor configuração é fornecida pelo algoritmo genético [13].

É necessário destacar, no entanto, que é possível que problemas não triviais resolvidos por AGs tenham a necessidade de um alto poder computacional para sua resolução. Assim como em outros casos mais gerais, uma alternativa para o aumento de desempenho de algoritmos genéticos pode ser a aplicação de técnicas de distribuição. Isso é necessário, sobretudo, pela ineficiência atingida na execução de AGs em computadores de propósito geral disponíveis no mercado. Além da restrição em relação ao poder computacional encontrado em um processador comercial, outros fatores também favorecem a distribuição neste caso. A possibilidade de utilizar implementações diferentes nos procedimentos internos do AG, em relação a versão serial, é uma vantagem bastante interessante e atrativa [12], [7], [1], [5]. Desse modo, a segunda parte deste trabalho compreende a distribuição das simulações do AG, inicialmente implementadas de forma seqüencial. Os detalhes da distribuição serão apresentados em Sessões a seguir.

3 Otimização da Busca de Configuração

Um dos principais atrativos da ferramenta SimpleScalar é a capacidade de configurar praticamente todos os recursos de uma arquitetura. Por outro lado, essa grande quantidade de parâmetros usados para configurar a execução da simulação gera um grande número de configurações possíveis. Isso impossibilita a simulação de todas elas com a finalidade de escolha da melhor.

O processo de obtenção de uma configuração consiste em executar uma simulação, extrair e analisar os resultados. Os resultados são usados para gerar uma nova configuração, que é simulada, recomeçando o processo novamente. Este processo termina somente quando uma configuração que atenda as necessidades do usuário ou do projeto é alcançada ou quando o resultado obtido é considerado o melhor possível.

Este trabalho propõe uma ferramenta que, através da implementação de um algoritmo genético, automatiza o processo de obtenção de uma configuração, conforme descrito anteriormente. Em ambos os casos, o resultado encontrado não tem garantia de ser o melhor possível, mas sim, o melhor resultado obtido através da simulação de um determinado número de simulações.

O funcionamento da execução de uma geração do algoritmo genético está ilustrado na Figura 1. Cada uma das configurações da arquitetura é representada por indivíduos do algoritmo genético. Inicialmente, um nodo central gera a população inicial de indivíduos, ou seja, de configurações. A partir daí, o nodo central envia as configurações para que os demais nodos façam o processamento das simulações. O cálculo do *fitness* é obtido através da execução do simulador, que gera os resultados que são usados no cálculo da média de desempenho das simulações. Este desempenho é calculado pela análise de valores dos campos presentes nos resultados das simulações, em conjunto com um valor de peso definido na configuração da arquitetura simulada. Este peso é usado para obter uma configuração que represente o compromisso do projeto, seja ele referente ao custo, área ou outros fatores.

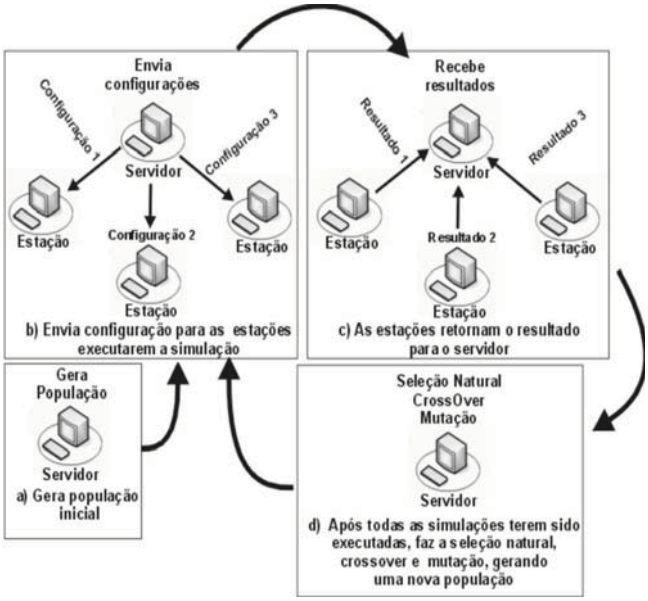

Figura 1. Funcionamento da execução de uma geração do algoritmo genético

Depois que o *fitness* de cada indivíduo foi obtido, é realizada a seleção natural para permanência dos mais aptos. O *crossover* e a mutação são usados para gerar uma nova população. Este processo se repete pela quantidade de gerações definidas pelo usuário na inicilização do sistema.

A execução das simulações realizadas pelo algoritmo genético pode ser seqüencial ou paralela pela distribuição em grade [6] ou agregado [4]. Os resultados de uma

simulação também são armazenados para uma eventual e provável consulta posterior, visto que é comum que alguns conjuntos de simulações sejam repetidas. Isso é aplicável quando uma ou mais simulações são executadas a partir uma mesma arquitetura, *benchmark*, dados de entrada do *benchmark* e quantidade de instruções executadas, o resultado deve ser igual.

Os dados da configuração de uma arquitetura também são armazenados para identificar se a simulação foi executada anteriormente. Estas informações são armazenadas em um banco de dados SQL para facilitar as consultas e obtenção dos relacionamentos entre os dados coletados.

4 Configuração do Algoritmo Genético

Além do controle da execução e resultados, a ferramenta apresenta uma interface bastante amigável para a configuração do ambiente de simulação. A Figura 2 mostra a tela principal, que consiste nos parâmetros de configuração da simulação e do AG propriamente dito. Também é possível configurar a execução dos *benchmarks* com seus respectivos dados de entrada, além do banco de dados responsável por armazenar e carregar resultados previamente executados.

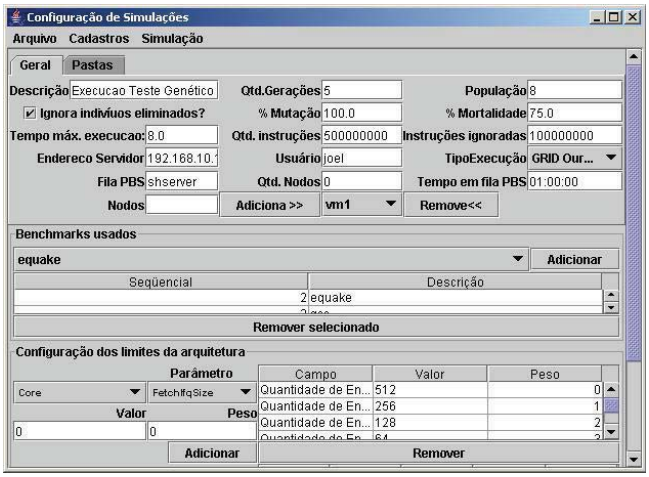

Figura 2. Tela de Configuração do AG

É possível observar que praticamente todos os parâmetros podem ser ajustados, entre eles: a quantidade de gerações do algoritmo genético (Qtd. de gerações); a população do algoritmo genético; a eliminação dos indivíduos após a seleção natural quando a nova população é composta (Ignora indivíduos eliminados); percentual de mutação e mortalidade (% Mutação e % Mortalidade); *benchmarks* e limites da arquitetura, que indica o valor usado para o parâmetro do *sim-outorder* e seu peso relacionado, podendo ser definidos um valor mínimo e máximo para o parâmetro, o peso inicial e a variação do peso, que será somada ao peso inicial (Configurações para os limites da arquitetura).

5 Metodologia de Simulação

Para avaliar os resultados da ferramenta, um caso típico no estudo e avaliação de arquitetura de microprocessadores foi escolhido como base. Este estudo de caso, investigado sob vários aspectos relevantes, foi necessário, sobretudo, para definição dos resultados ideais, usando busca exaustiva.

Este estudo típico diz respeito à cache nível 1 e 2 de instruções. Os parâmentros referentes a esse tipo de memória foram variados conforme os padrões atuais e simulados como originalmente seriam. Os valores usados nessas configurações estão descritos na Tabela 1 e, basicamente, exercem variação na capacidade da memória através dos seus respectivos números de bloco, larguras de bloco e associatividades. Os demais parâmetros utilizados neste trabalho correspondem a uma arquitetura superescalar atual e não sofreram nenhuma modificação ou variação ao longo das simulações aqui apresentadas. A utilização dos parâmetros que variaram geraram trezentas e noventa e nove configurações diferentes.

Tabela 1. Parâmetros utilizados para configuração das arquiteturas simuladas

Parâmetros	Configuração
Capacidade total da memória	16 KB, 32KB, 64KB e 128 KB
Número de blocos	32, 64, 128, 256 e 512
Tamanho do bloco	32 e 64
Associatividade	1, 2, 4 e 8

Cada uma das trezentas e noventa e nove configurações foram simuladas usando cinco *benchmarks* de inteiros do conjunto SPEC CPU2000 [9] tipicamente utilizados na avaliação de processadores atuais. Cada simulação executou quinhentos milhões de instruções e os resultados foram contabilizados após ignorar as primeiras cem milhões. Isto é feito para que os trechos iniciais dos programas, normalmente responsáveis pela inicialização de áreas de memória, não interfiram nos resultados.

Os *benchmarks* usados foram o *bzip2, equake, gcc, parser* e *vpr*. Assim sendo, para a obtenção dos primeiros resultados exaustivos, quase duas mil simulações foram executadas. Seus resultados foram então avaliados e a solução considerada ótima foi escolhida para servir de comparação com os resultados obtidos através do AG.

6 Resultados

Para avaliar a eficiência da ferramenta proposta, é necessário que a qualidade da configuração obtida pelo algoritmo genético seja próxima à configuração ótima, mas com um menor tempo de execução. Sendo assim, a qualidade da configuração foi medida através da execução do algoritmo genético e o resultado obtido pelo algoritmo genético foi comparado ao melhor resultado obtido pela busca exaustiva. Após, os resultados do tempo de execução do AG genético em relação à execução regular das simulações foram analisados. Finalmente, o ganho de desempenho obtido através da distribuição do processamento do próprio algoritmo genético usando a ferramenta OurGrid [2] e um agregado de computadores.

6.1 Qualidade dos Resultados Produzido pela Busca com Algoritmo Genético

A qualidade da configuração obtida pelo algoritmo genético foi avaliada comparando o resultado da configuração ótima, conseguida através dos experimentos discutidos da Seção 4, e o resultado obtido pelo algoritmo genético.

Com os resultados das mil novecentas e noventa e nove simulações realizadas, o *fitness* de cada configuração foi calculado usando os pesos definidos de acordo com as informações da Tabela 2. Os pesos definidos pelo usuário tem função essencial na obtenção de uma arquitetura que represente o melhor custo x benefício, ou seja, o melhor desempenho com um menor custo. O valor do peso possibilita a ferramenta calcular o valor do custo da arquitetura e o desempenho é obtido através da simulação. Estes pesos são definidos pelo usuário do momento da configuração da ferramenta, e para as simulações da memória cache nível 1 e 2 de instruções, os valores dos pesos são correspondentes à área ocupada pela memória cache. Quanto menor o valor do peso, maior deverá ser a área ocupada.

Tabela 2. Parâmetros utilizados para configuração das arquiteturas simuladas

Parâmetros	Peso				
	0	1	2	3	4
Quantidade de entradas	512	256	128	64	32
Tamanho do bloco	64	32			
Associatividade	8	4	2	1	

A partir deste ponto, a execução das simulações foi realizada através do AG. Neste caso, a configuração usada no algoritmo genético foi de uma população com oito indivíduos, executando por cinco gerações. Essa configuração foi escolhida já que a utilização de muitos indíviduos e gerações aumentaria ainda mais o tempo de simulação, e não há indicações de que isso resultaria em um ganho significativo na qualidade do resultado. O percentual de mortalidade e o percentual de mutação utilizados foram de 75% e 100%, respectivamente, maximizando a geração de novas arquiteturas para simulação nas gerações futuras.

A Figura 3 mostra os resultados gerados pela execução do AG. O eixo horizontal mostra as várias gerações de indivíduos que foram criados ao longo da execução. O eixo vertical, por sua vez, mostra o desempenho da arquitetura através da relação entre o IPC (Instruction Per Cycle) e o custo da configuração, de acordo com o peso informado pelo usuário. É possível verificar que, já a partir da segunda geração, os resultados matém-se bastante semelhantes.

Figura 3. Resultado da execução do algoritmo genético

Ao fim da execução do Algoritmo Genético, a ferramenta armazena os resultados em uma base de dados relacional, possibilitando ao usuário realizar consultas SQL. Os melhores resultados alcançados pela execução de todas as configurações estão dispostos conforme a Tabela 3. Além do valor do resultado ainda estão presentes a ordem do resultado, assim como o código identificador da configuração utilizada na simulação que atingiu aquele resultado. Como discutido anteriormente, este resultado, produzido pelo AG, é obtido através da relação entre a média do IPC e o custo da configuração. Isso significa que a ferramenta faz uma média do IPC alcançado por uma determinada configuração executada por todos os *benchmarks* e relaciona com o valor do custo da configuração. É válido ressaltar que a ferramenta leva em conta os pesos previamente estabelecidos na escolha destes resultados para efetuar uma escolha mais justa. Isso ocorre porque dependendo do compromisso do projeto, a escolha não se dará pelo resultado mais alto e sim pela combinação do custo x benefício.

De acordo com o cálculo efetuado pelo AG e apontado pela ferramenta, a melhor configuração foi a de código 236, que se refere à uma memória cache de instruções com 128 entradas, tamanho do bloco de 32 bits e associatividade 4. O índice alcançado representa 98,48% da melhor configuração considerada ótima através da simulação exaustiva das trezentas e noventa e nove configurações.

Porém, essa configuração considerada ótima também aparece entre as melhores colocadas na execução do AG e corresponde ao código de configuração 139, referindo-se à uma memória cache nível 1 com 64 entradas, tamanho do bloco de 64 bits e associatividade 4 e cache nível 2 com 32 entradas, tamanho do bloco de 64 bits e associatividade 8.

Tabela 3. Melhores resultados da execução de todas as configurações

Ord	Conf	Valor	Ord	Conf	Valor	Ord	Conf	Valor
1	139	1.8659	2	230	1.8570	3	174	1.8551
4	233	1.8551	5	168	1.8549	6	169	1.8540
7	167	1.8538	8	171	1.8536	9	175	1.8531
10	162	1.8531	11	181	1.8517	12	126	1.8485
13	131	1.8469	14	127	1.8466	15	136	1.8456
16	123	1.8448	17	144	1.8444	18	214	1.8442
19	215	1.8403	20	140	1.8385	21	224	1.8382
22	236	1.8374	23	229	1.8363	24	243	1.8355

6.2 Redução do Tempo de Execução Através do AG

Essa Seção destina-se a apresentar os valores correspondentes ao ganho de desempenho, considerando-se o tempo de execução, da busca pela melhor configuração através do AG e a busca exaustiva. Desse modo, todas as simulações executadas tiveram os seus tempos totais tomados, para que fosse possível fazer esse cálculo.

O tempo de execução de todas as trezentas e noventa e nove simulações foi obtido através da multiplicação entre a quantidade de simulações executadas e a média do tempo de execução de todas as simulações. Cada simulação registrou o tempo de início e fim de sua execução, possibilitando a obtenção da média de tempo de execução das simulações. O valor obtido da média do tempo de execução de uma simulação foi de dezoito minutos. Como o uso do algoritmo genético reduziu a quantidade de simulações executadas, o valor da média do tempo de execução foi usado para ilustrar a diferença entre o tempo gasto na execução das simulações do AG e de todo o espaço de busca. Usando a média do tempo de execução, é possível calcular o tempo aproximado de execução de todas as simulações que foi de vinte e quatro dias, vinte e duas horas e trinta minutos.

Similarmente, para calcular o tempo de execução do algoritmo genético é necessário multiplicar a quantidade de simulações realizadas pelo tempo de execução de cada simulação. O algoritmo genético na primeira geração de indivíduos executa quarenta simulações, pois para cada indivíduo são executadas cinco simulações, sendo uma para cada *benchmark*. Nas quatro gerações seguintes, a quantidade de simulações executadas diminui em cinco, porque o indivíduo mais apto permanece na geração seguinte e as informações dos resultados da sua simulação estão presentes no banco de dados, não necessitando executá-la novamente. Desse modo, foram realizadas pelo algoritmo genético cento e oitenta simulações, com tempo calculado de dois dias e seis horas.

O tempo de execução do algoritmo genético representa 9,02% do tempo de execução de todas as trezentas e noventa e nove possibilidades de configuração, significando um ganho de 90,98%.

6.3 Distribuição das simulações do AG

Para diminuir o tempo de execução do algoritmo genético, a execução das simulações foi realizada paralelamente usando as tecnologias de grade e agregado. O ganho de desempenho era esperado, mas era necessário quantificá-lo. Nos testes realizados para medir o ganho de desempenho de uma execução distribuída, as configurações do algoritmo genético não foram alteradas. Além disso, cada nodo do agregado e grid possui capacidade de processamento e memória similares ao utilizado no caso seqüencial.

Para essa parte dos experimentos, duas abordagens foram usadas com a finalidade de testar o desempenho da distribuição em mais de uma situação. Essas abordagens foram criadas através da variação do número de *benchmarks* utilizados. A primeira situação é quando o número máximo de simulações simultâneas é menor ou igual à quantidade de computadores disponíveis para a distribuição. A segunda situação é quando a quantidade máxima de simulações simultâneas é maior que a quantidade de computadores disponíveis. O número máximo de simulações simultâneas é igual ao número máximo de simulações realizadas em uma geração do AG, calculado pela multiplicação entre o número de indivíduos da população e o número de *benchmarks*.

Assim como na Seção anterior, a hora de início e fim de cada simulação realizada no agregado foi armazenada possibilitando calcular o tempo médio da execução de uma simulação. Da mesma forma, o tempo de execução seqüencial foi calculado pela multiplicação entre a quantidade de simulações executadas pelo algoritmo genético e o tempo médio da execução de uma simulação.

As informações sobre o primeiro teste realizado no agregado estão na Tabela 4. A primeira parte da tabela mostra a configuração do agregado, assim como as características deste primeiro grupo de experimentos. A segunda parte da tabela, por sua vez, mostra os resultados obtidos. As próximas tabelas apresentadas nessa Seção também seguem essa organização.

Nessa primeira situação, o processamento de setenta e duas simulações foi distribuído em dezesseis nodos. A comunicação entre os processos foi implementada usando *sockets*. Em cada geração do algoritmo genético o número

Tabela 4. Dados sobre o primeiro teste realizado em um agregado

Característica	Configuração
Número de nodos usados	16
Número de benchmarks	2
Número máximo de simulações por geração	16
Dados	**Resultados**
Tempo médio da execução de uma simulação	00:24:30
Número de simulações executadas	72
Tempo de execução seqüencial	29:24:00
Tempo de execução do AG com distribuição	02:54:07
Percentual de ganho	90,13%

máximo de simulações por geração não ultrapassa a quantidade de nodos disponíveis. Portanto, cada nodo executa uma simulação por geração. O tempo total da execução distribuída foi 90,13% menor, se comparada à execução seqüencial do algoritmo genético.

As informações sobre o segundo teste realizado em um agregado estão na Tabela 5.

Tabela 5. Dados sobre o segundo teste realizado em um agregado

Característica	Configuração
Número de nodos usados	20
Número de benchmarks	5
Número máximo de simulações por geração	40
Dados	**Resultados**
Tempo médio da execução de uma simulação	00:24:30
Número de simulações executadas	180
Tempo de execução seqüencial	73:30:00
Tempo de execução do AG com distribuição	05:05:58
Percentual de ganho	93,06%
Ganho sobre busca exaustiva	99,37%

Nessa segunda situação, a configuração do AG que foi distribuída é a mesma usada para avaliar a qualidade de busca do resultado, inclusive o número de *benchmarks* simulados. Isso foi realizado para tornar possível uma comparação justa entre a obtenção de uma configuração por busca exaustiva, algoritmo genético com execução seqüencial e algoritmo genético com distribuição de processamento.

O processamento de cento e oitenta simulações foi distribuído em vinte nodos. Em cada geração do algoritmo genético o número máximo de simulações por geração é o dobro da quantidade de nodos disponíveis. Portanto, cada nodo pode executar mais de uma simulação por geração. O tempo total da execução distribuída foi 93,06% menor, quando comparada à execução seqüencial do algoritmo genético. Se comparada à execução da busca por força exaustiva, o ganho no tempo de execução é de 99,37%.

Os testes utilizando grade foram realizados em um laboratório com 16 máquinas disponíveis, usando a ferramenta OurGrid no sistema operacional Debian Linux. Os resultados obtidos no primeiro conjunto de simulações em grade estão descritos na Tabela 6, enquanto que o segundo conjunto encontra-se na Tabela 7.

Tabela 6. Dados sobre o primeiro teste realizado usando grade

Característica	Configuração
Número de computadores usados	16
Número de benchmarks	2
Número máximo de simulações por geração	16
Dados	**Resultados**
Tempo médio da execução de uma simulação	00:14:56
Número de simulações executadas	72
Tempo de execução seqüencial	17:55:12
Tempo de execução do AG com distribuição	01:33:28
Percentual de ganho	91,30%

As características desse conjunto de testes foram idênticas aos primeiros, submetidos no agregado. Usando a ferramenta OurGrid, o tempo total da execução distribuída foi 91,30% menor se comparado à execução seqüencial do algoritmo genético, apresentando um pequeno ganho se comparado à distribuição no agregado.

Tabela 7. Dados sobre o segundo teste realizado usando grade

Característica	Configuração
Número de computadores usados	8
Número de benchmarks	2
Número máximo de simulações por geração	16
Dados	**Resultados**
Tempo médio da execução de uma simulação	00:14:56
Número de simulações executadas	72
Tempo de execução seqüencial	17:55:12
Tempo de execução do AG com distribuição	02:44:59
Percentual de ganho	84,66%

Já que a grade disponível possui 16 máquinas apenas e a segunda situação prevê que em cada geração do algoritmo genético o número máximo de simulações por geração é o dobro da quantidade de nodos disponíveis, a quantidade de simulações foi mantida e o número de nodos disponíveis para a distribuição foi diminuído pela metade. Nesse caso,

a distribuição em grade ainda obteve 84,66% de ganho de desempenho se comparada à execução seqüencial.

Com isso é possível concluir que tanto no caso do agregado quando no caso da grade, a utilização de distribuição das simulações realizadas pelo AG apresenta ganhos significativos, quando comparados à versão seqüencial. É importante observar, no entanto, que mesmo a versão seqüencial obteve uma grande redução no tempo de busca à melhor configuração, se comparado à busca exaustiva.

7 Conclusões

A utilização de simulação em projetos de microprocessadores modernos é uma realidade. Mesmo sendo uma boa alternativa, encontrar uma configuração ótima não é uma tarefa trivial, tendo em vista o enorme número de recursos que podem ser variados. Tipicamente, milhares de simulações devem ser executadas e analisadas. Este trabalho apresenta uma alternativa para este problema, utilizando um Algoritmo Genético para otimizar a busca pela configuração ótima. Apesar de ser uma heurística, o AG pode atingir um desempenho bastante significativo e chegar muito próximo dos resultados obtidos por uma busca exaustiva. Assim sendo, a ferramenta apresentada aqui pode substituir a morosa tarefa de geração de configurações, execução das simulações e avaliação de resultados, já que a mesma realizada todas essas etapas de uma forma comprovadamente eficiente.

Considerando os resultados da validação do algoritmo genético, o tempo de execução diminuiu 91% e a qualidade da configuração obtida foi muito próxima da ótima, com resultados acima de 97%, usando como parâmetro o melhor resultado.

Em relação à distribuição das simulações realizadas pelo AG, em todos os testes e formas de distribuição, o resultado do ganho em tempo de execução teve resultados muito bons, com valores superiores a 80%.

No caso dos testes realizados usando o agregado, o segundo teste obteve um resultado ligeiramente melhor que o primeiro, porque usou quatro nodos a mais. Isso ocorreu apesar de ter a quantidade de nodos disponíveis igual à metade da quantidade máxima de simulações por geração.

O ganho em desempenho obtido pelo aumento da quantidade de computadores fica mais evidente pela comparação entre os resultados dos testes usando distribuição em grade. Nestes testes, onde somente a quantidade de nodos é alterada, o ganho em desempenho diminui significativamente quando a quantidade de nodos é diminuída pela metade.

Finalmente, é interessante destacar que a ferramenta aqui proposta pode ser facilmente integrada a qualquer simulador derivado do *sim-outorder*. Se os parâmetros de entrada e características de saída forem iguais, a integração é imediata. Nos demais casos, alguns ajustes simples podem ser rapidamente efetuados com a mesma finalidade.

Agradecimentos

Os autores agradecem o suporte oferecido na forma de bolsas e auxílios da UNISC. Adicionalmente, os autores agradecem a toda a equipe do Laboratório de Tecnologia em Clusters DELL da UFRGS.

Referências

[1] P. Adamidis. Review of parallel genetic algorithms bibliography. Technical report, Internal Tech. Report, Automation and Robotics Lab., Dept. of Electrical and Computer Eng., Aristotle University of Thessaloniki, Greece, 1994.

[2] N. Andrade, L. Costa, G. Germóglio, and W. Cirne. Peer-to-peer grid computing with the ourgrid community. In *Proceedings of the SBRC 2005 – IV Special Tools Session*, 2005.

[3] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, 1997.

[4] R. Buyya and C. Szyperski. *Cluster Computing*. Nova Science Publishers, 2001.

[5] E. Cant'u-Paz. A summary of research on parallel genetic algorithms. Technical Report IlliGAL report 95007, University of Illinois at Urbana-Champaign, 1995.

[6] W. Cirne and et. al. Computing for bag of tasks applications. In *Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Goverment*, 2003.

[7] J. R. Filho, C. Alippi, and P. Treleaven. Genetic algorithm programming environments. In J. Stender, editor, *Parallel Genetic Algorithms: Theory and Applications*, pages 65–83. IOS Press, Amsterdam, 1993.

[8] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program analysis. *Journal on Instruction-Level Parallelism (JILP)*, (7), Sept. 2005.

[9] J. L. Henning. SPEC CPU200: Measuring CPU performance in the new millenium. *IEEE Computer*, 33(7):28–35, July 2000.

[10] J. H. Holland. Adaptation in natural and artificial systems. Technical Report Report of the Systems Analysis Research Group SYS–1/92, University of Dortmund, Department of Computer Science. Ann Arbor, MI: The University of Michigan Press, 1975.

[11] M. Olivieri. A genetic approach to the design space exploration of superscalar microprocessor architectures. In *ISCAS (5)*, pages 69–72. IEEE, 2001.

[12] C. B. Pettey, M. R. Leuze, and J. J. Grefenstette. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 155–161, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[13] A. Thompson. Artificial evolution in the physical world. In T. Gomi, editor, *Evolutionary Robotics: From intelligent robots to artificial life (ER'97)*, pages 101–125. AAI Books, 1997.

[14] G. V. R. Viana. *Meta-heurísticas e programação paralela em otimização combinatória*. UFC Edições, Fortaleza, 1998.