1
Introdução

Antes de surgirem as primeiras linguagens de programação, computadores eram programados instrução por instrução usando-se o sistema binário ou hexadecimal. Esta tarefa era cansativa e sujeita a muitos erros. Programas eram difíceis de se ler e modificá-los era mais difícil ainda porque todos os programas tinham que ser escritos usando-se endereços absolutos. Computadores caríssimos ficavam ociosos por longos períodos enquanto programas eram desenvolvidos, o que tornava o custo do desenvolvimento de software muito alto.

A primeira linguagem de programação de alto nível foi FORTRAN (FORmula TRANslation). Criada em 1954 pela IBM, FORTRAN foi desenvolvida especificamente para a comunidade científica. Naquela época os programas tinham não mais que 400 linhas de código e computadores eram máquinas utilizadas apenas por laboratórios de pesquisa e grandes corporações como empresas aéreas e governos federais. Somente em 1956, pesquisadores do MIT iniciaram experimentos com entrada direta via teclado em computadores. Ao longo do tempo, surgiram várias outras linguagens e ferramentas de programação, visando facilitar o desenvolvimento de software. Ainda na década de 50, surgiram as linguagens Lisp e COBOL.

Em meados da década de 60 o computador começou a ficar mais popular. Minicomputadores eram vendidos a um custo razoável para pequenas empresas e laboratórios científicos. As linguagens Simula, Algol68, Pascal e Basic surgiram nesta época. Mas somente em 1974 surgiu a primeira estação de trabalho com mouse embutido, janelas gráficas, menus, ícones e a possibilidade de conexão com uma rede local, embora esta máquina nunca tenha sido vendida comercialmente. Foi nesta época que foram criadas as linguagens Prolog e C.

A comunicação computador/computador expandiu-se quando, em 1970, o departamento de defesa americano estabeleceu 4 nós na chamada ARPANET: a Universidade da Califórnia em Santa Bárbara, UCLA, a SRI International e a Universidade de Utah. A ARPANET foi criada para ser uma rede de compartilhamento

de recursos abrangente. Seus projetistas propuseram-se a atingir várias metas: uso direto de serviços de hardware distribuídos, recuperação direta de dados em bancos de dados remotos e o compartilhamento de subrotinas de software e pacotes não disponíveis no computador primário do usuário. Em 1973, Robert Metcalfe idealizou o método Ethernet de conexão de redes, no centro de pesquisa da Xerox, em Palo Alto, que mais tarde se tornaria um padrão para conexão de redes locais.

Na década de 80, a IBM revolucionou o mercado com o lançamento do PC (*Personal Computer*) com o sistema operacional DOS da Microsoft, disparando o rápido crescimento do mercado de computadores pessoais. Logo em seguida, a Apple lançou o Macintosh, o primeiro computador de sucesso com uma interface gráfica e mouse, que deu início a uma nova gama de aplicações como softwares para desenho que fazem uso do mouse e editores de texto que usam a técnica de processamento de texto WYSIWYG (*What You See Is What You Get*). Linguagens como Ada e C++ surgiram nessa época.

Foi em 1985, que surgiu o conceito de linguagem de coordenação com a criação de Linda, uma linguagem para processamento paralelo, que usa um modelo de memória compartilhada. Foi no mesmo ano que a Internet moderna ganhou apoio, quando a NSF (National Science Foundation, EUA) formou a NSFNET, conectando cinco centros de super computadores. Logo, várias redes regionais se desenvolveram, até que o governo americano redesignou partes da ARPANET à NSFNET.

Mais tarde, no início da década de 90, surgiram linguagens como Darwin, chamadas linguagens de configuração, criadas para descrever estruturas de software de forma declarativa.

Em 1990, a linguagem de marcação de hipertexto HTML (HyperText Markup Language) deu origem a *World Wide Web*. Usando especificações como URL (Uniform Resource Locator) e HTTP (HyperText Transfer Protocol), a HTML colaborou fortemente com a expansão da Internet.

A popularização da Internet e o baixo custo dos computadores levaram ao desenvolvimento de novas aplicações e de centenas de novas linguagens. Preocupações com o desempenho das linguagens ganharam uma nova perspectiva e a portabilidade de aplicações passou a ser muito importante.

A idéia de se compartilhar programas entre diferentes plataformas de hardware para poupar tempo e custo no desenvolvimento de programas já havia surgido há muitas décadas. Naquela época, foram projetadas famílias de máquinas com sistemas compatíveis, que compartilhavam um conjunto comum de instruções. Uma vez que o software era compatível entre toda uma família de máquinas, houve um grande

aumento no uso de programas compartilhados. Entretanto, a existência de diferentes plataformas de hardware tornava inviável que todas elas fossem unificadas.

Após a grande expansão da Internet, as preocupações com o compartilhamento de programas entre plataformas heterogêneas e o estabelecimento de um ambiente unificado de programação e computação levaram ao aparecimento de novas linguagens de programação, como Java.

Com a disponibilidade de máquinas com poder de computação cada vez maior, linguagens de script executadas por um interpretador tornaram-se uma tendência importante na Internet. No final da década de 80 e início da década de 90, surgiram linguagens como Perl, Tcl, Python, JavaScript e Lua.

Com o baixo custo do hardware e o aumento de sua velocidade, tornou-se atraente pagar o preço de se ter uma linguagem mais flexível, mesmo que ela seja menos eficiente. Não que a eficiência de uma linguagem não seja mais importante. A eficiência sempre será uma preocupação constante entre os projetistas de linguagens. Por exemplo, em redes geograficamente distribuídas, o ambiente de execução de uma aplicação distribuída é altamente dinâmico, o que torna particularmente interessante o uso de linguagens mais flexíveis.

Outra grande mudança que promoveu inovação em linguagens de programação foi a computação paralela e distribuída, que vem se tornando cada vez mais viável devido ao baixo custo do hardware e do rápido desenvolvimento de redes de computadores. Há uma demanda de novas linguagens de programação que dêem suporte a este tipo de atividade de computação. Atualmente existem duas abordagens para esta questão. Uma é adaptar as linguagens de programação disponíveis, como C e FORTRAN, e provê-las com bibliotecas especiais. O resultado são linguagens paralelas e distribuídas como CC++ [1], pC++ [2] e HPF (High Performance Fortran) [3] ou o aparecimento de bibliotecas de comunicação como PVM [4] e MPI [5]. Outra abordagem é projetar uma linguagem conceitualmente nova, que explore os princípios fundamentais da computação paralela e distribuída. O resultado são linguagens de programação como Linda [6], Orca [7] e SR [8].

Ao escolher uma linguagem de programação para um novo projeto, várias coisas devem ser levadas em consideração. Uma questão importante é a facilidade de uso da linguagem, ou seja, será mais fácil e rápido programar nela do que outras linguagens? Em 1960, COBOL provavelmente era mais fácil que as outras opções disponíveis, mas novas linguagens de programação e novos paradigmas mudaram os conceitos do que se considerava fácil de se usar.

O domínio de aplicações para o qual a linguagem foi projetada também é

importante. Para escrever programas científicos, até hoje muitos pesquisadores usam FORTRAN, mas para escrever uma aplicação Web, Java é uma linguagem muito mais popular.

O desempenho de uma linguagem também permanece uma questão importante. É um dos motivos pelos quais FORTRAN e C ainda serão utilizados por muitos anos. Mas algumas aplicações não precisam de uma linguagem de alto desempenho. Por exemplo, em programas interativos, o desempenho da linguagem provavelmente não vai afetar o resultado final da aplicação, pois, em muitos casos, o tempo de resposta esperado pelo usuário é compatível com o desempenho oferecido por linguagens menos eficientes.

O projeto de um sistema de computação é um equilíbrio entre o custo e desempenho do hardware e linguagens e técnicas de programação. A tendência atual de linguagens de programação é sacrificar parte da eficiência para obter programas mais fáceis de se ler, mais modulares e mais reusáveis.

Porém, aparentemente, nenhuma linguagem sozinha ou arquitetura computacional seria capaz de lidar satisfatoriamente com todas os aspectos de desenvolver uma aplicação complexa e multifuncional. Além disso, problemas como reusabilidade, composicionalidade e extensibilidade se tornaram de vital importância.

Para lidar com todos esses requisitos de aplicações de larga escala, apareceram as noções de programação multilinguagem ou multiparadigma [9]. Existem basicamente duas formas de se oferecer mecanismos de programação multiparadigma ou multilinguagem: projetar uma nova super linguagem, que oferece as facilidades de todos os paradigmas que se deseja usar ou oferecer uma interface entre linguagens existentes.

1.1 A Tese Proposta

Neste trabalho defendemos a tese de se usar um modelo multilinguagem de programação para aplicações geograficamente distribuídas.

Em programação paralela, a necessidade de lidarmos com problemas tais como heterogeneidade e tolerância a falhas, especialmente ao lidar com ambientes de memória distribuída, levou ao desenvolvimento de modelos de programação multilinguagem [10]. Muitos pesquisadores também identificaram a necessidade de separarmos interesses e requisitos da computação propriamente dita e da cooperação

e comunicação entre componentes computacionais. Modelos de coordenação [10, 6] oferecem esta separação de interesses, freqüentemente propondo linguagens distintas para programar esses dois tipos de atividades.

Há alguns anos atrás, quando se falava em programação paralela distribuída, estávamos nos referindo a computações em máquinas paralelas fechadas, clusters de computadores ou ainda redes locais. Neste cenário, o ambiente de execução de aplicações era bastante estável, seus recursos estavam sempre disponíveis e não havia uma grande preocupação com segurança ao se executar um programa. Havia um interesse muito maior em configurar essas aplicações com parâmetros iniciais do que em adaptá-las a mudanças no ambiente de execução.

Atualmente, devido ao crescimento das redes geograficamente distribuídas e às iniciativas de projetos de computação em grade [11, 12], a programação paralela distribuída vem ganhando força neste ambiente de execução altamente dinâmico. Apesar dos desafios que este ambiente impõe, tais como configuração altamente heterogênea e dinâmica, variação de recursos disponíveis e questões de segurança, o poder computacional que se pode obter, a um custo relativamente baixo, é extremamente compensador.

Enquanto o desempenho de programas paralelos e distribuídos é uma grande preocupação quando se usa ambientes "fechados" (clusters e redes locais), uma vez que o tempo de comunicação de uma aplicação é tipicamente pequeno comparado ao tempo de processamento que ela faz, em ambientes "abertos" (redes geograficamente distribuídas), a flexibilidade e a possibilidade de se adaptar a aplicação em tempo de execução são muitas vezes considerados mais importantes.

Neste trabalho decidimos explorar o uso de um modelo de programação multilinguagem, que já vínhamos utilizando como uma ferramenta de programação paralela em ambientes de memória distribuída [13], como um modelo de coordenação.

ALua [14, 15] (Asynchronous Lua) é um modelo de comunicação assíncrono, orientado a eventos, que usa a linguagem de extensão Lua [16, 17]. No meu trabalho de mestrado [13], concluímos que a perda de desempenho observada em ambientes fechados não era grande e que este modelo era viável. Como parte desta tese, investimos na sua infra-estrutura e, em [18, 14], fizemos uma extensão deste modelo que permite o processamento concorrente de fluxos de dados e de controle entre processos. Agora estamos explorando a flexibilidade que este modelo oferece para construir mecanismos de adaptação, de modo a facilitar a alteração do comportamento de uma aplicação durante sua execução.

Usamos a flexibilidade que o ALua oferece, como um modelo de coordenação,

para alterar o comportamento de uma aplicação durante sua execução. Estudamos seu uso para monitoramento e coordenação de aplicações com diferentes requisitos, como aplicações paralelas e aplicações multimídia, e seu uso em diferentes ambientes, como um *cluster* e uma grade.

Nosso objetivo não foi reimplementar aplicações existentes, mas prover técnicas para introduzir mecanismos de monitoramento e adaptação, interferindo o mínimo possível (ou tanto quanto se queira) no código existente. Quanto mais código ALua, mais flexibilidade, mas o programador deve decidir o grau de "intrusão" em seu código.

Alguns trabalhos sobre adaptação têm que prever com antecedência as mudanças que podem ocorrer na aplicação. Segundo [19], esse tipo de reconfiguração dinâmica, onde mudanças são previstas em tempo de desenvolvimento, pode ser classificado como *reconfiguração programada*. Por outro lado, na *reconfiguração ad-hoc*, ou *não planejada*, as ações de reconfiguração são estabelecidas pelo usuário em qualquer momento durante a execução da aplicação, como é o caso do ALua. Com o ALua a decisão de quando ou o por que adaptar e como adaptar uma aplicação pode ser tomada após a aplicação ter sido disparada, sem que seja preciso interromper sua execução para que estas adaptações sejam feitas.

Através do uso de uma linguagem de extensão como Lua, uma das inovações desse trabalho é trazer ao domínio de aplicações paralelas distribuídas o papel de um usuário administrador que pode não só monitorar o ambiente e suas aplicações, mas também interferir na execução de uma aplicação durante sua execução.

Neste trabalho, avaliamos as contribuições que o modelo utilizado pelo ALua pode trazer para o desenvolvimento de aplicações paralelas distribuídas, como um modelo de coordenação de aplicações, oferecendo uma ferramenta para desenvolvimento e coordenação de aplicações e monitoramento do próprio ambiente de execução.

1.1.1 Modelos de Coordenação

Modelos de desenvolvimento de software baseados em coordenação oferecem uma separação clara entre componentes de software individuais e suas interações. Essa separação, juntamente com o maior nível de abstração geralmente oferecido por modelos e linguagens de coordenação, aumenta a produtividade do desenvolvimento de software, facilita a manutenção, melhora a modularidade, promove reusabilidade

e leva a organizações e arquiteturas de software que são mais fáceis de se controlar, verificar e analisar.

O uso de modelos de coordenação mostra-se relevante no projeto, desenvolvimento, depuração, manutenção e reuso de sistemas distribuídos. Acreditamos que um modelo de coordenação flexível, com fronteiras menos rígidas entre os componentes da aplicação, é primordial no contexto de sistemas abertos, sistemas com entidades móveis e sistemas de evolução dinâmica e reconfigurável.

1.1.2 O Modelo do ALua

ALua é um sistema para programação de aplicações paralelas em ambientes de memória distribuída, baseado num modelo multilinguagem de programação. ALua usa a linguagem de extensão Lua [16, 17] para coordenar a interação entre componentes escritos em C. Como outros ambientes distribuídos, uma aplicação em ALua é composta por um grupo de processos executando em vários computadores e comunicando-se através de uma rede. Código Lua cuida da comunicação entre os processos (e conseqüentemente define a arquitetura da aplicação), enquanto funções C cuidam das tarefas de uso intensivo da CPU, em cada processo.

Uma característica importante do uso de uma linguagem interpretada é a provisão de um mecanismo para a execução de trechos de código criados dinamicamente. No ALua, mensagens são trechos de código que são executados pelo receptor. Isto oferece um mecanismo de comunicação muito simples, no entanto muito poderoso. Existe apenas uma primitiva de comunicação, `send`, que envia um trecho de código para outro processo. Não existe o equivalente a uma primitiva `receive`. Ao invés disso, ALua usa um modelo de programação dirigido a eventos, onde a chegada de uma mensagem é tratada como um evento. Este mecanismo de comunicação é bastante flexível: Um programador pode usá-lo trivialmente para tarefas simples, como chamar uma função remota, mas ele também pode usá-lo para tarefas muito mais complexas, como trocar remotamente o algoritmo que um processo está executando. No contexto de aplicações paralelas de longa duração, e com a disponibilidade de um console interativo, esta é uma possibilidade poderosa, e permite ao programador redefinir dinamicamente o comportamento da aplicação.

1.2 Organização do Texto

No próximo capítulo discutimos o conceito de coordenação. Falamos de modelos, linguagens e da importância do uso de coordenação em sistemas paralelos distribuídos.

No Capítulo 3, apresentamos o modelo básico do ALua. Mostramos exemplos de programação e discutimos a flexibilidade que se pode obter através de seu uso.

No Capítulo 4, apresentamos uma extensão do ALua para tratamento de múltiplos canais de comunicação e exemplos de uso deste mecanismo.

No Capítulo 5, mostramos como o ALua pode ser usado em aplicações paralelas para redes geograficamente distribuídas, dando suporte ao monitoramento e à adaptação dinâmica destas aplicações.

Finalmente, no Capítulo 6, apresentamos alguns trabalhos relacionados, as principais contribuições desta tese e trabalhos futuros.