UMA LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA AMBIENTES PARALELOS.

Laís do Nascimento Salvador
Líria Matsumoto Sato

Laboratório de Sistemas Integráveis
Edificio da Engenharia da Eletricidade
Escola Politécnica - Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, nº 158
05508-900 - São Paulo, SP
Tel: (011) 818-5270/818-5589
e-mail: lnsalvad@lsi.usp.br

RESUMO

Este artigo tem como objetivo apresentar o projeto de uma linguagem para codificação de programas paralelos, que utiliza o paradigma de orientação a objetos. Esta linguagem denominada ÁGATA apresenta tanto características do paradigma de orientação a objetos assim como primitvas para a expressão do paralelismo. Com este projeto pretende-se apresentar ao programador uma ferramenta eficiente e que facilite a programação de máquinas paralelas.

ABSTRACT

This article presents the project of a language for parallel programs that enforces the object-oriented paradigm. This language is called ÁGATA offers both the facilities of the object-oriented paradigm as well as primitives for the expression of parallelism in programs. This project intends to present an eficient tool that makes programming parallel machines easier.

1. INTRODUÇÃO

Com o avanço na área de computação de alto desempenho e a utilização crescente de múltiplos processadores são imprescindíveis linguagens, compiladores e ferramentas adequadas ao desenvolvimento de programas para computadores com arquiteturas paralelas.

São necessárias linguagens que permitam ao usuário explorar explícita ou implicitamente o paralelismo presente no programa. Linguagens têm sido propostas nos paradigmas de programação procedural, funcional, lógico e orientada a objetos. Uma explanação sobre estas linguagens e paradigmas encontra-se em [Mey88], [Sha86], [Hud86], [Bal89], [Geh92] e [Weg92]. O paradigma de programação orientada a objetos não apresenta a acurácia matemática presente nos paradigmas lógicos e funcional que proporciona a estas duas linhas uma grande fonte de paralelismo implícito [Dan88]. Por outro lado o encapsulamento das informações em objetos facilita a exploração do paralelismo no programa. Os atributos do objeto são acessados apenas via mensagens enviadas ao objeto, isto é, através de chamadas aos seus métodos, provendo segurança no acesso aos dados e confinando no objeto a análise de dependencias de dados.

No projeto de uma linguagem de programação para ambientes paralelos tem-se dois objetivos principais: o alto desempenho e oferecer facilidades para explorar o paralelismo implicita ou explicitamente.

Buscando atingir estes objetivos surgiu a proposta da linguagem ÁGATA [Sal94]. Nesta linguagem tem-se a fusão dos aspectos de paralelismo e da programação orientada a objetos, resultando em uma linguagem que oferece recursos para a geração de código eficiente para sistemas com múltiplos processadores e memória compartilhada, mantendo as vantagens oferecidas pelo paradigma de programação orientada a objetos, provendo modularidade e extensibilidade.

Neste trabalho é apresentada a linguagem ÁGATA e descritos alguns aspectos do sistema de programação e processamento.

2. CARACTERÍSTICAS DA LINGUAGEM

As principais características da linguagem ÁGATA são:

- Como os aspectos de paralelismo estão unificados com os conceitos de OO, o paralelismo é explicitado através dos objetos. O paralelismo, aqui, não é obtido através da criação de um processo para cada objeto [Wyat92]. As chamadas dos métodos são executadas assincronamente por um conjunto de processos, promovendo o paralelismo.
- As construções sintáticas que expressam os conceitos do paradigma de orientação a objetos seguem o modelo da linguagem C++ [Eck89], não sendo porém a ÁGATA uma extensão da C++ .
- Em uma classe não é permitida a definição de dados públicos. É apenas permitida a definição de dados privados que são acessados somente pela chamada aos métodos públicos da classe.
- Na linguagem ÁGATA não é permitida a utilização de variáveis globais, isto é, variáveis visíveis a todos objetos. As únicas estruturas de dados do programa são os próprios objetos e as variáveis locais dos métodos. Podem ser atribuídos a estas variáveis os valores de retorno das chamadas aos métodos.
- É permitido os uso de funções, que podem ser usadas no auxílio da programação.
- As construções sintáticas básicas como: atribuições, cálculo de expressões, definições de tipos (que não sejam classes), declaração e chamadas a funções, seguem o modelo da linguagem Ansi C [Ker90].
- A definição de uma classe está ligada à definição de um tipo de dados. Logo toda classe é um tipo, porém nem todo tipo é uma classe. Existem, na ÁGATA, tipos de dados que não são classes como: int, char, float etc., que foram herdados da linguagem C.

Como foi dito anteriormente os aspectos de paralelismo estão embutidos na semântica dos conceitos de orientação a objetos. Na ÁGATA pode-se explorar o paralelismo em três níveis:

- Paralelismo Inter-Objetos - um método de um objeto executando em paralelo com um método de outro objeto.
- Paralelismo Intra-Objeto - este paralelismo é interno a um objeto e pode ser subdividido em dois casos:
  - Paralelismo Inter-Métodos - um método executando em paralelo com outro método do mesmo objeto.

* Paralelismo Intra-Método- este paralelismo ocorre na execução de um método e é obtido através de chamadas assíncronas a outros métodos do mesmo objeto e de outros objetos. Também está previsto no projeto da linguagem primitivas para explicitar paralelismo em "loops" dentro dos métodos e posteriormente incluir a detecção automática desse tipo de paralelismo.

Estas três formas tratam de diferentes escopos e aspectos semânticos dentro da programação orientada a objetos (inter e intra objetos). Porém estes diferentes níveis de paralelismo se baseiam na premissa básica de que a chamada a qualquer método de qualquer objeto é assíncrona, isto é, a execução do método chamado é paralela à execução do método chamador. O paralelismo pode ser restrito em função da necessidade de sincronização criada por:

- acesso exclusivo aos atributos de um objeto;
- a necessidade de se esperar por um valor retornado por um método.

Estes aspectos serão melhor discutidos no item referente à sincronização.

3. DESCRIÇÃO DA LINGUAGEM

Como foi dito anteriormente, a ÁGATA é uma linguagem que segue o modelo da linguagem C++ no que diz respeito aos aspectos de orientação a objetos, onde o paralelismo é efetivado principalmente através de chamadas assíncronas aos métodos.A linguagem dessa forma torna-se muito simples a nível de construções básicas. Nesse tópico serão descritos as construções sintáticas da linguagem como também aspectos de sincronização e comunicação.

3.1 A declaração de classes

```
[monitor] class nome_classe {
[private:]
      Dados Privados ou Atributos;
// Métodos Privados
      [mutex] Método_Privado_i ( );
      ...
[public:]
// Métodos Públicos
      [mutex] Método_Público_i ( ); }
```

Foi adotada da linguagem C++ a utilização das palavras reservadas **public** e **private** para indicar os métodos públicos e a parte privada de cada classe. Os métodos públicos são aqueles que são visíveis aos usuários dos objetos da classes e são acessados através do envio de mensagens aos objetos. Na ÁGATA não foi permitida a definição de dados públicos, isto é, dados visíveis aos usuários do objeto, como acontece na C++, pois esta abordagem não segue os padrões do paradigma adotado. Além disso anularia uma das suas principais características: o encapsulamento de dados, que foi usada como argumento para a escolha deste paradigma em um ambiente paralelo.

A parte privada diz respeito à definição dos dados privados (atributos) e dos métodos privados. Os dados privados constituem o estado de cada objeto da classe. Eles não são acessados diretamente, somente através de chamadas aos métodos públicos. Os métodos privados são invisíveis aos usários do objeto e podem apenas ser chamados pelos métodos públicos daquela classe, provendo assim mais um nível de segurança.

As palavras reservadas **monitor** e **mutex** estão relacionadas com o acesso exclusivo aos atributos dos objetos. Quando é especificada a palavra reservada **monitor** significa que os objetos desta classe vão ser do tipo monitor, isto é, todos os métodos tem acesso exclusivo sobre os atributos do objeto e consequentemente estes métodos não podem ser executados em paralelo com nenhum outro método do mesmo objeto.

A palavra reservada **mutex** indica que aquele método tem acesso exclusivo sobre os atributos do objeto e não pode ser executado em paralelo com os outros métodos daquele mesmo objeto. Especificar a palavra reservada **monitor** tem o mesmo efeito de declarar todos os métodos da classe como **mutex**. Estes aspectos serão abordados no item referente à sincronização.

3.2 Declaração de objetos

Na declaração é utilizado o nome da classe como um tipo de dados e a seguir os nomes dos objetos daquela classe.

```
nome_classe nome_objeto_1,nome_objeto_2,..,nome_objeto_n;
```

Nesta fase, o compilador se encarrega de realizar duas tarefas:

- Quando é declarado um objeto é criada uma área de memória compartilhada para os seus atributos e um apontador para o conjunto de métodos da classe daquele objeto.
- Também na declaração, é chamado o método construtor correspondente àquela classe.

3.3 Chamadas aos métodos

Depois de declarado um objeto, são realizadas chamadas aos seus métodos. Como foi dito anteriormente todas estas chamadas são feitas de forma assíncrona. Na chamada é especificado o nome do objeto seguido do nome do método a ser chamado.

```
nome_objeto.nome_metodo_i(parâmetros);
```

As chamadas aos métodos do tipo **mutex** ou aos métodos de objetos do tipo **monitor** também são realizadas de forma assíncrona.

3.4 Especificação dos métodos .

A especificação dos métodos diz respeito à codificação do corpo de cada método. Ela é feita indicando-se o nome da classe a qual pertence o método seguida do nome do método em questão.

```
nome_classe::nome_método_i (parâmetros)
{

}
```

Está previsto no projeto da linguagem uma primitiva, para explicitar paralelismo a nível de loops, denominada **forall**. Ela pode ser usada na especificação dos métodos para aumentar o paralelismo no programa.

3.5 Aspectos de Sincronização

A sincronização é um dos aspectos mais problemáticos da programação paralela, pois se não for bem gerenciada pode vir a gerar situações desastrosas como: deadlock, dados inconsistentes e outras mais. No caso da ÁGATA, todas as chamadas aos métodos são assíncronas, porém há a necessidade de prever o retorno de chamadas a métodos e também de gerenciar o acesso exclusivo às variáveis compartilhadas entre métodos de um mesmo objeto.

Se não houvesse essas restrições, a explanação sobre a linguagem AGATA já estaria completa, isto é, uma linguagem paralela que utiliza conceitos do paradigma de orientação a objetos onde o paralelismo é efetivado principalmente através de chamadas assíncronas aos métodos dos objetos. Essa definição não deixa de ser correta como uma definição geral, mas não serve como uma caracterização completa da ÁGATA, pois omite certos aspectos que dizem respeito à sincronização. Para explicar melhor estes aspectos, são apresentados dois subitens: acesso exclusivo às variáveis compartilhadas e chamadas aos métodos.

Acesso exclusivo às variáveis compartilhadas:

Um dos pontos relevantes a ser considerado na programação paralela é quando mais de um processo tem acesso às mesmas variáveis compartilhadas. Como a execução destes processos a princípio é paralela, um problema que pode ocorrer é a execução da alteração (escrita) destas variáveis compartilhadas em paralelo com outro processo de leitura ou escrita a estes dados. Esta situação pode gerar inconsistência nos dados compartilhados. Uma solução para este problema é não permitir a execução paralela de um processo escritor com qualquer outro processo quer de escrita ou de leitura sobre os mesmos dados compartilhados.

Em busca desta solução chega-se à questão da exclusão mútua, isto é, um processo que altere (escreva sobre) dados compartilhados deve ter acesso exclusivo sobre estes dados excluindo assim os outros processos (de leitura/ escrita) de acessarem a estes mesmos dados, durante o tempo da alteração. Porém um processo leitor pode estar sendo executado em paralelo com outros processos leitores sobre os mesmos dados compartilhados. Esta abordagem do problema resulta num ambiente de execução em que em cada instante há apenas um processo escritor executando sobre um determinado conjunto de dados compartilhados ou vários processos leitores executando em paralelo sobre este conjunto de dados [Cha90].

Como foi dito anteriormente, na linguagem ÁGATA são permitidos os seguintes níveis de paralelismo: o paralelismo inter-objetos e o paralelismo intra-objeto.

No caso do paralelismo inter-objeto, no que diz respeito ao acesso exclusivo aos atributos dos objetos não há problemas pois cada objeto contém seus próprios atributos que são alterados pelos seus

próprios métodos. Dois métodos de objetos diferentes podem executar em paralelo sem problemas com relação ao acesso a variáveis compartilhadas.

Quanto ao paralelismo intra-objeto que é subdividido em paralelismo inter-métodos e intra-métodos, sincronizações podem ser necessárias.

No primeiro caso, onde métodos do mesmo objeto são executados paralelamente tem-se o problema da exclusão mútua, pois os métodos têm acesso aos mesmos dados compartilhados que são os atributos dos objetos. Quando um método altera o estado do objeto é necessária a exclusão mútua, devendo então ser especificado como um método exclusivo. A especificação da exclusividade é feita através da palavra reservada **mutex** na declaração do método. Quando todos os métodos de uma classe são exclusivos, pode-se usar na declaração da classe a palavra **monitor**, indicando que todos os métodos dos objetos daquela classe são do tipo **mutex**.

O segundo caso diz respeito ao paralelismo intra-método, que é efetivado principalmente através de chamadas assíncronas a outros métodos, depara-se com o problema relacionado à liberação de exclusividade do acesso aos atributos do objeto. Isto ocorre quando na especificação do corpo dos métodos do tipo **mutex**, às vezes é necessária a chamada de outros métodos do mesmo objeto ou testar o valor de variáveis compartilhadas que são alteradas por outros métodos. Como a execução de um método **mutex** bloqueia a execução dos outros métodos do mesmo objeto, é necessário que este método libere este acesso exclusivo para permitir a execução de outros métodos do mesmo objeto. Em função disso a linguagem oferece uma primitiva de sincronização chamada **wait**. Esta primitiva providencia condicionalmente a liberação de exclusividade. O método **mutex** fica solicitando continuamente a exclusividade e liberando-a, até que a condição seja verdadeira.

Chamadas aos métodos:

Outra questão relevante no aspecto de sincronização diz respeito aos valores retornados de chamadas a métodos. Como na linguagem ÁGATA o paralelismo é obtido através das chamadas assíncronas aos métodos, depara-se com um problema que é quando um destes métodos retorna um valor. Uma primeira solução seria fazer esta chamada de forma síncrona, já que ela retorna um resultado e assim a execução do método chamador fica bloqueada "esperando" este resultado. Porém esta solução iria reduzir sensivelmente o paralelismo. Para se resolver este problema, na ÁGATA

adotou-se a seguinte postura: todas as chamadas são realizadas de forma assíncrona, independentemente do retorno de um valor. No caso de uma chamada retornar um valor, quando este valor for reutilizado, a execução do método chamador ficará bloqueada até que o valor atualizado fique disponível, isto é, até o término da execução do método chamado.

Ex.:

```
class x {
        ...
public:
        ...
int metodo1();
int metodo2();
        ...
}
void main()
{
x  A;
int i,k;
        ...
P1      k  =  A.metodo1();
        ...
P2      i = k;
        ...
}
```

A chamada em P_1 é realizada de forma assíncrona, tendo o metodo1() executado paralelamente ao main. Em P_2 execução main é bloqueada, esperando a finalização da execução do metodo1() e o retorno do resultado . Dessa forma o paralelismo é explorado ao máximo só sendo a execução bloqueada quando for necessário o resultado retornado pelo método solicitado.

No projeto do compilador da linguagem ÁGATA está prevista a geração desse mecanismo de sincronização de forma automática. O usuário, nesse caso, não precisa especificar nenhum tipo de primitiva de sincronização.

3.6 Comunicação

Como na ÁGATA não há variáveis compartilhadas entre objetos, a comunicação é possível através de argumentos de métodos e valores de retorno.

3.7 Exemplo

```
class matriz {
   double a[500][500], b[500][500], c[500][500];
public:
    matriz( );
    void mult( );
}
matriz::matriz ( )
{
   int i,j;
   for (i=0; i<size; i++)
      for (j=0; j<size; j++){
          a[i][j] = (double) i+j;
          b[i][j] = (double) i+j;
          c[i][j] = (double) i+j;
      }
}
void matriz::mult ( )
{
   int i,j,k,cont;
   for (cont=0; cont<100; cont++) {
      for (i=0; i<size; i++)
         for (j=0; j<size; j++){
            a[i][j] = (double) i+j;
            b[i][j] = (double) i+j;
                  c[i][j] = (double) 0.0;
         }
      for (i=0; i<size; i++)
         for (k=0; k<size; k++)
            for (j=0; j<size; j++)
               c[i][j] = c[i][j] + a[i][k]*b[k][j];
   }
}
void main ( )
{
   matriz M1, M2, M3, M4, M5, M6, M7, M8;

   M1.mult( );
   M2.mult( );
   M3.mult( );
   M4.mult( );
   M5.mult( );
   M6.mult( );
   M7.mult( );
   M8.mult( );
}
```

Neste exemplo como a classe **matriz** possui apenas dois métodos: o método construtor **matriz** e o método **mult**, este último não precisa ser declarado como **mutex**. No método **main** foram declarados oito objetos do tipo matriz e foram realizadas oito chamadas assíncronas ao método **mult**.

4. SISTEMA DE PROGRAMAÇÃO E PROCESSAMENTO

O compilador da linguagem ÁGATA está sendo implementado usando as ferramentas de geração automática de compiladores FLEX e BISON [Don90]. Na geração de código paralelo pelo compilador está sendo usada uma outra linguagem paralela chamada CPar [Sat92]. Já foram implementadas as funções do ambiente de execução que foram codificadas em CPar.

Nesse tópico serão descritas algumas abordagens de implementação que foram adotadas neste projeto e apresentados alguns resultados obtidos em termos de desempenho.

4.1 Escalonamento

Em várias abordagens há uma associação direta entre um objeto e um processo [Wya92]. Nesta linha, para cada objeto criado é criado um processo correspondente que executa os métodos do objeto. Porém seguindo este caminho pode ocorrer alguns problemas como a criação excessiva de processos e a limitação da quantidade de processos criados.

A estratégia adotada neste trabalho é a implementação de um servidor de processadores virtuais para a execução dos métodos dos objetos. Uma abordagem similar é usada em [Cha90]. O servidor funciona com uma fila, sendo que cada método chamado é inserido nessa fila à espera de um processador livre para executá-lo. Para cada processador físico disponível para execução do programa, cria-se um processo correspondente (processador virtual). Estes processadores virtuais consultam a fila e retiram da fila os métodos para serem executados. Com esta estratégia, em um determinado instante há no máximo "n" processos executando em paralelo, onde "n" é o número de processadores disponíveis da máquina.

Usando este esquema de escalonamento, resolvem-se os problemas do "overhead" na criação de processos, evitando ultrapassar o limite de processos criados, como também aproveita-se o máximo de paralelismo.

4.2 Problemas com "deadlock"

A possibilidade de ocorrência de "deadlock" é um problema relevante que deve ser estudado, pois trata-se de um ambientes paralelo, onde existem: processos compartilhando recursos, uso de

semáforos, regiões críticas, enfim problemas típicos de programação paralela. Será então descrito uma situação detectada em que pode ocorrer esse tipo de problema e a solução encontrada.

Uma situação em que pode vir a ocorrer "deadlock" é quando há sucessivas chamadas a métodos encadeadas, isto é, um método chamando outro método e o número de processadores disponíveis para a execução do programa é inferior ao número de chamadas encadeadas. Para ficar mais claro suponha que um método que está sendo executado chame um outro método. O método chamado é inserido na fila e o método chamador continua sendo executado. O problema pode acontecer se há apenas um processador disponível para a execução deste programa e o método chamador necessite em algum instante do resultado da execução do método chamado. Nesse caso irá ocorrer "deadlock", pois o método chamador ficará bloqueado "esperando" o término da execução do método chamado e este por sua vez não vai ser executado pois o único processador disponível está executando o método chamador.

Esta situação de "deadlock" pode acontecer quando ocorrem as seguintes condições:

- "n" métodos estão sendo executados e cada um deles chama um outro método.
- cada um dos "n" métodos chamadores, em algum instante durante a sua execução, precisa "esperar" o término da execução do método chamado.
- há apenas "n" processadores virtuais disponíveis para a execução do programa.

Para evitar este tipo de problema, será adotada a seguinte solução: associar um estado a cada chamada realizada. A chamada de um método pode assumir os seguintes estados:

1 - o método se encontra na fila de execução;

2 - o método está sendo executado por algum processador;

3 - o método já foi executado.

No ponto em que o método chamador necessitar do resultado da execução de algum método chamado assincronamente, vai ser verificado o estado daquela chamada.

Se ela se encontra no estado 3 a execução do método chamador pode continuar pois o método chamado já foi executado.

Se ela se encontra no estado 2, o método chamado já está sendo executado por outro processador logo não há risco de "deadlock", então a execução do método chamador fica bloqueada à espera do fim da execução do método chamado.

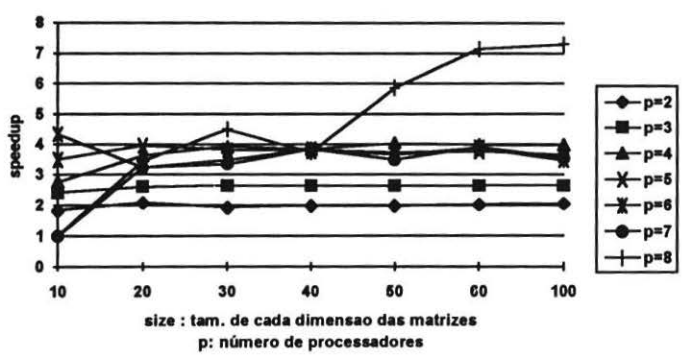
Se ela se encontra no estado 1, a execução do método chamador não deve ficar bloqueada, pois uma situação de "deadlock" pode ocorrer. Nesse caso o processador, que está executando o método chamador, irá executar o método chamado de forma síncrona, através de uma chamada de função em C.

Este mecanismo também resolve o problema da finalização de um programa em ÁGATA. O programa em ÁGATA está finalizado quando as seguintes condições são satisfeitas simultaneamente:

- a fila de execução está vazia
- não há métodos no estado 2
- o método main termina

4.3 Análise de Desempenho

Foram feitas algumas simulações para o exemplo do item 3.7. Estas simulações foram feitas na máquina Silicon Graphics que possui oito processadores com memória compartilhada. Foram realizados vários testes com 2,3, até 8 processadores com o programa paralelo e calculado o "speedup" com relação ao programa sequencial correspondente. Os seguintes resultados foram obtidos:



A partir da análise desses dados observa-se que os melhores resultados aparecem nos testes em que o número de processadores é divisor do valor oito. Isto deve-se ao fato que neste exemplo são executados oito métodos de forma assíncrona. Nos testes em que o número de processadores não são

múltiplos de oito, alguns processadores ficam desocupados, em função do esquema de escalonamento. Nesses casos o "speedup" aproxima-se dos testes cujo o número de processadores seja imediatamente menor e divisor de oito. Dos testes cujo o número de processadores é divisor de oito, o melhor resultado obtido foi com dois processadores onde o "overhead" é menor na criação de processos ("speedup"=2). No caso com oito processadores os resultados melhoraram sensivelmente com o aumento de **size**, chegando a um "speedup" superior a sete com size igual a 60 e a 100. Isto se explica pois quando há uma criação de um número maior de processos é mais compensador executar programas maiores.

5. CONCLUSÃO

A linguagem ÁGATA, o compilador e o sistema de processamento são componentes do projeto do sistema ONIX, um ambiente para desenvolvimento de programas paralelos [Sat94], que se baseia no paradigma de programação orientado a objetos. A fusão dos aspectos de paralelismo e do paradigma de programação orientado a objetos mostrou bons resultados na definiição da linguagem ÁGATA. Proporcionando assim a exploração adequada dos recursos de processamento disponíveis no computador, comprovada pelos resultados obtidos na análise de desempenho, e mantendo as vantagens oferecidas pela programação orientada a objetos.

Na fase atual de projeto o compilador para a linguagem ÁGATA está sendo implementado. A parte referente às funções de escalonamento já foram implementadas. Pretende-se no futuro incluir na linguagem mecanismos de herança entre classes, característica importante do paradigma OO. Também está prevista no projeto do compilador a exploração de paralelismo em loops.

6. BIBLIOGRAFIA

[Bal89] - BAL,H.E., STEINER,J.G., TANENBAUM,A.S., **Programming Languages for Distributed Computing Systems**. ACM Computing Surveys, vol.21, no.3, setembro, 1989.

[Cha90] - CHANDRA,R.; GUPTA,A.; HENNESSY,J.L.; **COOL: A language for parallel programming**, In: Programming, Languages and Compilers for Parallel Computing. MIT Press, England, 1990.

[Dan88] - DANFORTH,S.; TOMLINSON,C.; **Type Theories and Object-Oriented Programming.** ACM Computing Surveys, vol.20, no.1, março, 1988.

[Don90] - DONNELLY,C.; STALLMAN,R.; **BISON The YACC - compatible Parser Generator.** Free Software Foundation, dezembro, 1990.

[Eck89] - ECKEL,B. **Using C++.** Osborne McGraw-Hill, 1989.

[Geh92] - GEHANI,N.H.; ROOME,W.D.; **Implementing Concurrent C.** Software-Practise and Experience, vol.22, março, 1992.

[Hud86] - HUDAK,P. **Para-Functional Programming**. Computer, agosto, 1986 .

[Ker90] - KERNIGHAN,B.W.; RITCHIE,D.M.; **C, A linguagem de programação.** Editora Campus, 2a. Edição, 1990.

[Mey88] - MEYER,B. **Object oriented software construction**. Prentice-Hall, Englewood Cliffs, NJ. 1988.

[Sal94] - SALVADOR,L.N., SATO,L.M.; **ÁGATA: Uma Linguagem de Programação Orientada a Objetos para Ambientes Paralelos.** Relatório Técnico - no.35, LSI-USP, março, 1994.

[Sat92] - SATO,L.M. **Um Sistema de Programação e Processamento para Sistemas Multiprocessadores.** In: Anais do VI Simpósio Brasileiro de Engenharia de Software, outubro, 1991.

[Sat94] - SATO,L.M.; HUZITA,E.H.M.; SALVADOR,L.N.; HSIANG,H.T.; **ONIX: An Environment for the Development of Parallel Object-Oriented Software.** In: Anals of the International Workshop on High Performance Computing: Compilers and Tools. março, 1994.

[Sha86] - SHAPIRO,E. **Concurrent Prolog: A Progress Report**. Computer, agosto, 1986.

[Weg92] - WEGNER,P. **Dimensions of object-oriented modeling**. Computer, oct 1992.

[Wya92] - WYAIT,B.B., KAVI,K., HUFNGEL,S. **Parallelism in object-oriented languages: a survey**. IEEE Software, pg. 56-66,nov. 1992.