2
Conceitos do Desenvolvimento de Aplicações Paralelas

Vamos abordar os conceitos envolvidos no desenvolvimento de aplicações para que seja possível propor uma visão em camadas a fim de abstrair cada nível de suporte ao paralelismo.

Começando pela arquitetura da máquina, onde o processador pode apresentar mais de um núcleo, até a linguagem de programação, vários aspectos influenciam no desenvolvimento e precisam ser conhecidos pelo programador para que possa tomar as decisões sobre como implementar seu algoritmo. O uso indevido acarretará em um desempenho baixo ou em alto custo de desenvolvimento, o que compromete o investimento em uma máquina ou ambiente. Consideraremos quatro aspectos que serão apresentados a seguir.

2.1
Arquiteturas Paralelas

O primeiro aspecto a ser abordado é o da máquina em si, isto porque podemos ter diferentes arranjos de máquinas para prover a facilidade de paralelismo. As máquinas paralelas podem ser generalizadas em quatro categorias:

1. processadores especializados, como os usados na década de 1970;
2. máquinas compostas de mais de um processador (de núcleo simples) interligados internamente e com acesso a memória compartilhada;
3. máquinas compostas por um processador com mais de um núcleo por pastilha, como os processadores multi-núcleo lançados comercialmente a partir de 2005;
4. máquinas, sejam de núcleos simples ou multi-núcleo, interligadas via rede, sem memória compartilhada, formando um *cluster*.

Podemos então pensar em mais de um processador por máquina e ao mesmo tempo mais de um núcleo por processador, além de poder interligar essas máquinas em rede juntamente com máquinas com processadores especializados tal como o Cell/BE, criando um *cluster* heterogêneo. Em nosso trabalho consideramos dois tipos de trabalho, um primeiro onde permitimos a

programação de uma máquina com processador multi-núcleo, e um segundo onde permitimos a programação de um *cluster* heterogêneo mantendo a mesma API do primeiro tipo.

Uma outra categoria bem mais recente de paralelismo, é o uso de aceleradores com vários núcleos de processamento para realizar parte da computação de uma aplicação, tais como as placas de vídeo massivamente paralelas. Essas placas foram, inicialmente, desenvolvidas para trabalhar com processamento de problemas relativos à computação gráfica, sendo posteriormente estendidas para trabalhar com problemas mais gerais (nVidia2007, Khronos2010). Embora esse tipo de arquitetura baseada em GPUs (*Graphical Processor Unit*) tenha um papel importante no desenvolvimento de aplicações paralelas, não faz parte do escopo desse trabalho devido a suas especificidades. Podemos, contudo, comparar a complexidade de programação do processador Cell/BE à de programação dessas placas sem o risco de comprometer a abstração que queremos propor.

2.2 Desenvolvimento baseado em Paralelismo e Distribuição

Se considerarmos o desenvolvimento de aplicações paralelas a partir das máquinas baseadas em *clusters*, veremos que este tipo de ambiente oferece uma alternativa de baixo custo para paralelismo, tornando a computação de alto desempenho disponível para uma maior quantidade de usuários. Para este tipo de ambiente, a abordagem mais comum é usar a passagem de mensagens como o meio de comunicação entre os processos que participam de uma dada computação. Então como segundo aspecto vamos considerar duas classes de técnicas de programação paralela:

1. Programação Procedural Tradicional
2. Programação Baseada em Abstrações

Por razões de clareza, ambientes que têm pouca ou nenhuma preocupação com os aspectos de engenharia de software são considerados um subgrupo da primeira classe, mesmo que os programas não sejam verdadeiramente estruturados.

2.2.1 Programação Procedural Tradicional

O paradigma mais comum e difundido é usar o *Message Passing Interface* (MPI) em um estilo estruturado procedural. Tanto operações *send/receive* quanto coletivas, que se concentram em grupos de processos, estão em uso.

Esta técnica é difundida devido às muitas bibliotecas de linguagens procedurais como C e Fortran, que são muito utilizadas em aplicações científicas. Devemos também considerar que muitas outras áreas da ciência, como Física e Química contam com profissionais que desenvolvem seus próprios modelos e programas de forma independente dos profissionais de engenharia de software como visto em (Gropp1999) e (Danis2006).

Estes cientistas-programadores tendem a aprender programação paralela de forma *on-the-job* e na maioria das vezes se recusam a incorporar boas práticas de programação em seu trabalho diário. Quando fazem isso, acabam ficando longe de seus objetivos de pesquisa e tornam-se mais próximos de um programador de domínio específico. Como podemos ver em (Danis2006) às vezes é possível que cientistas e programadores trabalhem juntos, tanto como uma equipe ou em uma relação cliente / servidor.

2.2.2 Programação Baseada em Abstrações

O uso de objetos na programação paralela com C++ e Java foi inicialmente explorado através do encapsulamento do MPI em objetos, mas dependentes e ligados ao paradigma procedural estruturado. Notadamente, existem implementações C++ de MPI disponíveis, no entanto, devido à sua compatibilidade com a linguagem de programação C, as primitivas são usadas diretamente e não de forma orientada a objetos (Gropp1999).

Posteriormente, o paradigma de objetos começou a ser explorado com implementações MPI, tanto em C++ (Gropp1999), Java (Baker1999)(Mohamed2002), Ruby (Ong2002) e Python (Miller2002), quanto com ambientes paralelos que permitem a passagem de mensagens para os estilos de comunicação totalmente baseado em objetos. Nesses trabalhos foram exploradas técnicas de grupos de objetos e chamada a um método do grupo.

Nosso maior interesse recai sobre as técnicas que podem prover a gestão da complexidade e recursos de modelagem em conjunto com a separação de conceitos para os problemas maiores.

Um dos principais benefícios do uso de componentes é a possibilidade de ter grandes repositórios de software testado e garantido que funcione adequadamente e pode ser reutilizado (Sommerville2001). Essa característica aumenta a confiabilidade e reduz os riscos associados ao processo de software a ser desenvolvido.

Outra vantagem do modelo de componentes é ser baseado em execução em *containers*, responsáveis pela manipulação de interações entre componentes e ciclo de vida, mas que não são capazes de qualquer cálculo específico do apli-

cativo. Isso nos leva a um modelo em que o código funcional não está vinculado a questões de infraestrutura, sendo servido pelo *container*, quando necessário. Em nosso trabalho buscamos essa ideia de prover uma infraestrutura para a aplicação através de um conjunto de camadas sobrepostas, porém sem o uso de componentes de forma que a aplicação não precise ser desenvolvida de acordo com um modelo predefinido.

2.3 Linguagens Interpretadas no Paralelismo

O terceiro aspecto a ser considerado é o das ferramentas utilizadas diretamente pelo desenvolvedor, que são as linguagens de programação e seus ambientes de execução. Nosso interesse está nas linguagens interpretadas por acreditarmos que o uso de uma máquina virtual proporciona um nível de isolamento propício para aplicação de otimizações com mínimo impacto nas aplicações.

Vários autores têm apresentado alternativas para permitir o uso de linguagens interpretadas na área de computação científica e paralela devido a grande popularidade dessas linguagens. Em (Luszczek2007) é apresentada uma estratégia para permitir o uso de Python através de uma compilação transparente. Diferentemente de outros projetos baseados em JIT, o trabalho explora o conceito que chamamos aqui de **desempenho necessário**. A primeira avaliação realizada é se o desempenho atual está satisfatório. Caso não esteja, o código Python é submetido a um processo de compilação que gera um código em C correspondente à aplicação do usuário. Nossa proposta difere dessa abordagem porque acreditamos que mesmo com o uso da interpretação pura o desempenho necessário pode ser atingindo. Consideramos como alternativa o uso de uma máquina virtual Lua capacitada a executar compilação JIT, e além disso oferecemos o suporte a execução em *clusters* heterogêneos com uma interface de comunicação embutida e transparente para o usuário.

Outro trabalho que busca integrar o suporte a paralelismo em Python é apresentado em (Hinsen2006). Os autores apresentam um estudo de integração da biblioteca BSP para prover mecanismos de comunicação entre processos Python com o objetivo de fornecer uma melhor abstração na construção de aplicações paralelas mas mantendo o desempenho pela execução de código nativo integrado ao ambiente Python.

Uma abordagem que busca mais transparência é utilizar mecanismos de paralelização automática. Um exemplo dessa linha é apresentado em pR (Li2011) que não precisa que o código da aplicação seja modificado, porém pR é uma linguagem interpretada de propósito específico voltada para o pro-

cessamento de aplicações estatísticas. Como o ambiente provê um mecanismo automático de paralelização o programador não tem a visão de uma aplicação paralela, ao invés disso executa sua aplicação sequencial. A análise de dependência dos dados é feita pelo ambiente pR, baseado no padrão Master/Worker, que submete as chamadas de função e *loops* as unidades *worker*. Diferente de pR, acreditamos que a escolha das construções paralelas, bem como da análise de dependências, devem ser realizadas pelo desenvolvedor de aplicações paralelas. Embora exista essa divergência na visão da responsabilidade do desenvolvedor de aplicação, ambos os modelos convergem para o uso de linguagens interpretadas por oferecerem uma camada de abstração na qual os desenvolvedores de sistema podem trabalhar para a melhoria do desempenho de execução. Uma outra diferença importante é que em nossa abordagem temos diferentes máquinas virtuais executando, dessa forma podemos facilmente usar técnicas de JIT em cada uma das máquinas de forma independente dos mecanismos necessários para paralelização da execução. O código a ser executado é enviado para cada uma das máquinas virtuais que podem ser equipadas com JIT, assim somente as funções a serem executadas passam por um processo de compilação JIT diminuindo muito o custo de realizar essa tarefa.

Em (Parri2011) os autores argumentam que as linguagens interpretadas não conseguiram acompanhar a evolução dos processadores em relação aos novos tipos de opções para melhoria de desempenho, em especial às instruções do tipo SIMD. Embora a compilação para um formato binário intermediário permita alguma análise de oportunidades para emprego dessas instruções, esse alternativa se mostrou ineficiente em diversos cenários. Além disso o uso de técnicas de JIT não tem como fornecer os recursos computacionais necessários para uma identificação e vetorização de instruções. O trabalho propõe que as instruções SIMD sejam disponibilizadas para o desenvolvedor de aplicações na forma de bibliotecas para as linguagens interpretadas. Dessa forma o desenvolvedor pode expressar o emprego desse recurso eliminando a necessidade de uma análise por parte do compilador ou ambiente de execução, permitindo que sejam empregadas técnicas JIT apenas para gerar as chamadas que foram indicadas. A solução apresentada pelos autores utiliza os recursos de JNI para integração entre a máquina Java e o sistema nativo de forma que o desenvolvedor possa delegar a execução de partes mais lentas do seu código para a biblioteca. Em nosso modelo usamos uma abordagem semelhante através da biblioteca NativeArrays que permite representar um array em C pela aplicação Numina. Junto com essa biblioteca fornecemos funções para manipular esse array e principalmente para executar produtos vetoriais. Como essa função é compilada para a arquitetura destino o emprego de instruções

SIMD é realizado pelo próprio compilador.

De uma maneira geral o uso de linguagens interpretadas tem crescido bastante devido às facilidades que proporcionam em termos de velocidade de desenvolvimento, como apresentado em (Sankaralingam2010). No entanto nosso interesse está voltado para seu uso em desenvolvimento de aplicações paralelas como pode ser visto em (Gelernter1992), (Miller2002), (Ong2002), (Ururahy1999), (Hinsen2006), (Luszczek2007), (Williams2008), (McIlroy2010) e (Li2011).

Nos casos em que o desempenho atingindo for inferior ao necessário, algumas partes da aplicação podem ser escritas em alguma linguagem compilada, tipicamente em C, para prover uma melhora pontual sem comprometer a simplicidade e legibilidade do código. Neste cenário os mecanismos de integração com C apresentadas por Lua e Python são facilitadores para essa abordagem.

2.4 Padrões de Paralelismo

Como quarto aspecto vamos considerar uma visão mais recente do desenvolvimento de aplicações paralelas que é o emprego de padrões. Essa técnica, que já está consolidada no ambiente de aplicações sequenciais, vem sendo mais explorada no desenvolvimento de aplicações paralelas e já conta com um conjunto documentado de padrões tradicionais que podem ser encontrados em (Mattson2004).

O uso de padrões documentados só se torna possível quando temos um suporte adequado que permita ao desenvolvedor de aplicações pensar na infraestrutura como um conjunto de componentes ou serviços prontos para o uso, sem que seja necessário interagir com os diversos detalhes de cada ambiente. Neste contexto, buscamos em nosso trabalho fornecer os meios necessários para que seja possível utilizar esses padrões sem maiores dificuldades.

2.5 Como Quantificar o Desempenho

Para entender os ganhos reais de uma aplicação paralela torna-se necessário entender o que faz parte de uma avaliação de desempenho e custo. Para isso vamos utilizar como principais referências os modelos apresentados em (Andrews2000) e (Foster1995), onde os autores sugerem que o desempenho não pode ser considerado de forma isolada para avaliar uma solução paralela. Assim, quando quantificamos uma aplicação paralela torna-se necessário ana-

lisar outros fatores tais como eficiência do paralelismo, potencial para reuso, custos de hardware, portabilidade, e escalabilidade.

Mudar paradigmas de programação sempre traz dificuldades. No entanto, como a programação paralela é usada por uma certa classe de programadores que estão orientados para o desempenho, pode ser um pouco mais difícil expor as vantagens de usar níveis mais altos de abstração. Devemos, porém, considerar os benefícios que técnicas mais sofisticadas permitem em relação a melhor produtividade do programador e maior tempo de vida do software. Essa ideia é apoiada em diversos trabalhos (Catanzaro2011, Chafi2011, Luszczek2007, Sankaralingam2010, Tsoi2011, Vishkin2011).

A seguir apresentamos as principais métricas que, acreditamos, devem ser consideradas na quantificação de uma aplicação paralela.

2.5.1 Desempenho

A programação paralela é tradicionalmente centrada em aspectos como o tempo de execução e escalabilidade, e o desempenho de um sistema é frequentemente medido nesses termos. No entanto, os ganhos em desempenho podem ter um preço em termos de manutenção e evolução do código.

Como mencionado antes, uma série de parâmetros são definidos em (Foster1995) que devem ser considerados na definição do desempenho de um programa paralelo. Se considerar apenas o tempo de execução como métrica, o usuário terá uma simplificação excessiva e, muitas vezes, uma conclusão errada.

Em relação à abordagem para programação paralela, os autores de (Carriero1989) sugerem que deveríamos tentar desenvolver um programa paralelo de uma forma decomposta mais natural, por exemplo, alocar muitos nós de processamento para a computação. Se a aplicação não atingir o desempenho esperado, devemos percorrer o código aplicando técnicas de otimização – que por sua vez, tornam o código menos legível e de difícil manutenção – a fim de alcançar o desempenho necessário.

Devemos notar que há uma fronteira que terá de ser cruzada para atingir o limite de desempenho máximo, mas precisamos identificar se esse é o desempenho necessário para cada aplicação paralela. Na maioria dos casos, a solução de desempenho máximo quebra a estrutura do aplicativo (Kernighan1999), diminuindo sua legibilidade.

Ao explorar o desempenho de programas paralelos, é comum se referir ao desempenho máximo teórico que não é viável na prática, devido às limitações físicas do *hardware*. No entanto, para o escopo deste trabalho, consideramos

que o desempenho máximo absoluto é o limite superior de desempenho que pode ser alcançado na prática.

Podemos então diferenciar entre desempenho necessário e desempenho máximo de uma aplicação, considerando que o desempenho necessário é caracterizado pela demanda real do usuário enquanto o desempenho máximo é o limite físico que pode ser atingido por uma aplicação. Frequentemente o esforço necessário para atingir o desempenho máximo é considerável, já que para isso o desenvolvedor de aplicação precisará conhecer todos os detalhes da arquitetura usada. Uma questão que depende diretamente do problema que está sendo resolvido é a quantificação do desempenho necessário, mas podemos explicitar duas relações:

1. $\text{Desempenho}_{\text{Necessário}} \leq \text{Desempenho}_{\text{Máximo}}$
2. $\text{Esforço para Desempenho}_{\text{Necessário}} \leq \text{Esforço para Desempenho}_{\text{Máximo}}$

Ao usar abordagens baseadas em camadas de abstração, tais como as orientadas a objeto ou componentes, deve-se considerar o fato de que estas soluções usam um maior grau de indireção e chamadas de sub-rotina o que implica em perda de desempenho. No entanto, se o código de infraestrutura de paralelismo não se mistura ao da aplicação, é mais simples aplicar técnicas de otimização para ambas as partes sem comprometer a estrutura e a organização de cada uma. Também é importante notar que o código de infraestrutura deve ser otimizado por desenvolvedores de sistemas (Bentley2000, Gorlatch2004).

A pesquisa mostra formas de superar problemas de desempenho com a melhoria do ambiente de apoio, como em (Karwande2003) em que uma variante MPI é capaz de compilar código de comunicação de uma forma que melhora o desempenho para *clusters*. Por outro lado, em (Tan2003) e (Faraj2005), podemos ver uma abordagem baseada na geração de código para ambientes paralelos, o que permite atingir não só a melhora de qualidade e desempenho, mas também o melhor uso de recursos e da produtividade do programador. Uma versão modificada da biblioteca MPI é apresentada em (Ke2004) aplicando técnicas de compressão/descompressão as mensagens MPI, antes do envio e depois do recebimento, de forma totalmente transparente para o usuário. Todas estas abordagens podem ser empregadas a fim de permitir o aumento de desempenho da aplicação independentemente do programador. Essas técnicas de otimização podem ser embutidas em uma camada mais baixa e fornecidas ao desenvolvedor de aplicação de forma mais conveniente, facilitando sua tarefa.

2.5.2 Domínio de Aplicação

Conforme explicado em (Sommerville2001), domínio de aplicação é o comportamento global da aplicação para além das entidades específicas que estão sendo manipuladas. Isso pode levar a um modelo genérico para o processamento de diferentes instâncias do problema.

Este aspecto vai determinar se componentes de uma aplicação podem ser reutilizados em sua forma binária. Isto é importante porque, ao reutilizar, não queremos copiar o código-fonte de projeto para projeto. Além disso, o código binário, na maioria dos casos, foi testado em outros projetos, e provavelmente é mais confiável. De forma semelhante se a linguagem for interpretada podemos utilizar os módulos já prontos ou receber o código durante a execução de algum repositório.

Como podemos ver em (Szyperski2003), o uso de entidades de domínio específico não deve ser considerado de forma indiscriminada, pois isso poderá levar a um compromisso inicial com certas decisões de projeto que podem não satisfazer plenamente o problema. No entanto, se aplicada corretamente, esta abordagem pode melhorar significantemente o ciclo de desenvolvimento de software, fornecendo código binário, testado e pronto para usar que, na maioria dos casos, só tem de ser configurado para o novo problema. É importante notar que, por vezes, a configuração pode não precisar de recompilação de componentes, apenas de algum tipo de especialização.

Conforme relatado em (Matthey2004), o uso de uma maior abstração sobre uma estrutura orientada a objetos, para simulação de dinâmica de moléculas, ajudou os usuários a compreender as aplicações existentes e desenvolver novos algoritmos se concentrando nas especificidades desta tarefa, independentemente do suporte de infraestrutura necessária para executar o novo algoritmo.

Abordagens semelhantes foram utilizadas em (Jiao2003)(Gertz2003) e (Norton1995). Todos os autores consideraram que a utilização de técnicas de orientação a objeto no desenvolvimento de software, e o uso de herança permitiu maior reaproveitamento. Interfaces claras tornaram fácil a integração de módulos de equipes multidisciplinares e ajudaram a aumentar o nível de abstração, permitindo um desenvolvimento mais rápido de novas instâncias de problemas.

2.5.3 Complexidade

Complexidade do software é sempre um tema polêmico e de difícil abordagem. Conforme apresentado em (Evangelist1983), temos abordagens quantitativas para medir a complexidade computacional, mas isso pode ser enganador, uma vez que podem apontar alta complexidade em lógica fácil de compreender e, por outro lado, baixa complexidade em uma intrincada lógica que vai exigir maior esforço de compreensão. Embora as abordagens apresentadas em (Pressman1997) não estejam considerando as especificidades de desenvolvimento de software paralelo e distribuído, podem melhorar o desenvolvimento de software. Também em (Bhansali2005), o autor apresenta uma abordagem de relacionar o fluxo de controle com o de dados, o que parece ser mais realista, uma vez que considera o acoplamento de dados que pode estar presente. Além disso, se considerarmos o desenvolvimento de aplicações paralelas, a relação entre os dados e fluxos de controle é crucial, uma vez que na maioria dos casos este tipo de aplicação tende a processar grandes volumes de dados. Além disso, ter uma compreensão inicial da complexidade associada a um certo desenvolvimento pode orientar a equipe sobre as melhores escolhas.

Podemos considerar a complexidade como a quantidade de artefatos mais difíceis de implementar dos algoritmos que estão relacionados com o domínio a ser modelado, considerando as dificuldades associadas a um determinado algoritmo para o programador criar. Ao considerar as dificuldades associadas com as partes de comunicação, sempre poderemos recorrer à experiência de um programador especialista em infraestrutura, mas o entendimento completo dos problemas relacionados à própria aplicação vai exigir um grau maior de trabalho. Em última análise, poderíamos considerar que os padrões de comunicação – não importa quão difíceis – são repetidos na maioria dos casos apresentados, mas a aplicação que está sendo desenvolvida geralmente não pode se basear em algum modelo anterior.

Se estamos lidando com algoritmos difíceis, torna-se muito interessante aplicar técnicas de simplificação que poderiam ajudar a dividir o problema, para isso o uso de camadas, aumentando o nível de abstração, pode gerar simplificações graduais para o problema.

Isolando as partes do problema mais difíceis podemos simplificar a tarefa utilizando um especialista do domínio ajudando a escrever uma parte pequena de *software* sequencial e podemos, posteriormente, integrar esta solução à aplicação mais genérica.

2.5.4 As Opções Disponíveis

Várias opções foram apresentadas como diretrizes independentes, concentrando-se em cada possibilidade isoladamente, para compreendermos quais as alternativas podem ser usadas para endereçar grandes aplicações paralelas. Agora, estas opções são colocados juntas de maneira sistemática para tornar mais fácil sua compreensão.

Primeiro precisamos classificar as alternativas de implementação, que são apresentadas a seguir, em um grau crescente de abstração:

1. Ferramentas de alto desempenho aplicadas a programação procedural tradicional – a melhoria é obtida apenas no nível de desempenho com pouco ou nenhum avanço sobre a produtividade do programador.

2. Evolução dos programas existentes com componentes procedurais – esta alternativa pode proporcionar uma melhor produtividade quando o código legado começa a ser usado apenas como componentes de caixa-preta. O desenvolvimento é realizado em um modo misto entre procedural e baseado em objetos.

3. Uso de CORBA e MPI – aumenta drasticamente a produtividade, uma vez que permite a modelagem da aplicação separada da comunicação por uma camada, porém o código do aplicativo ainda está acoplado com o código de comunicação.

4. Uso de Camadas baseadas em MPI – uma abordagem semelhante à anterior, mas pode aproveitar a utilização de componentes de outros domínios para melhorar o desenvolvimento de software.

5. Uso de Camadas com Interfaces de Operações Coletivas – melhora de modelagem, separando os problemas de comunicação de forma que podem ser configurados e aplicados sem codificação explícita, mas através de escolha de políticas predefinidas.

6. Nova Linguagem de Programação Paralela – este é o mais alto nível de abstração possível, quando a própria linguagem incorpora os conceitos necessários para expressar todos os problemas recorrentes que são encontradas diariamente pelos desenvolvedores de aplicação.

Além dos níveis de abstração que foram apresentados, devemos considerar também as ferramentas de suporte que ajudam a proporcionar estes benefícios. Quatro itens apresentados podem ser incorporados ao desenvolvimento, pelo menos nos primeiros cinco níveis de abstrações. São eles:

1. A utilização de um mecanismo de compressão para troca de mensagens para melhorar o desempenho da rede – tal como apresentado em (Ke2004), esta é feita pela biblioteca de uma forma independente do programador.

2. Ajuste automático de operações coletivas com base na análise do *hardware* subjacente e padrão de comunicação – pode ser alcançada de uma forma estática ou dinâmica, pode promover a melhoria do desempenho global, ou pelo menos o mesmo resultado que o desenvolvimento manual, conforme visto em (Karwande2003), (Tan2003) e (Faraj2005).

3. O uso de otimizações do compilador – isso depende da disponibilidade do compilador, no caso de linguagens interpretadas aplicar otimizações na compilação da máquina virtual e/ou parâmetros para melhorar o desempenho da máquina virtual, ou até mesmo aplicar otimizações na forma como o código é interpretado.

Assim chegamos a um conjunto de diretrizes para o desenvolvimento de aplicações paralelas:

1. Separação de interesses para a evolução independente entre camadas – o desenvolvimento pode ser melhorado por ter camadas de prestação de serviços entre si, o que acaba por conduzir a melhorias independentes, que terão um impacto global.

2. Estimar as necessidades de desempenho reais – as vezes o código é otimizado muito cedo, consumindo esforço de desenvolvimento. Mais importante ainda, temos de assegurar que, quando a otimização ocorre, é executada nas partes de real importância do código e que proporciona um maior impacto. Gastar muito esforço para entregar uma pequena fração de melhoria de desempenho pode não ser uma boa ideia se o custo final for muito alto.

3. Estimar a carga de trabalho real – se a carga de trabalho do aplicativo não for considerada, podemos acabar com problemas de desempenho que dificilmente serão superados com técnicas simples. Será necessário quebrar todas as estruturas mal projetadas para atingir melhorias de desempenho que poderiam ser planejadas desde o começo.

4. Estimar a complexidade do software – se especialistas vão ser necessários, é melhor saber exatamente quais as partes que precisam de sua intervenção.

5. Identificar as oportunidades para isolar as entidades de domínio específico e processadores genéricos de algoritmo – este aspecto contribuirá para a criação de uma biblioteca de componentes para reutilização.

2.6 Conclusão

O desenvolvimento de aplicações paralelas tem alguns conceitos específicos que vão permitir lidar com as especificidades dessa disciplina. Cada um desses conceitos vai lidar com uma dimensão de paralelismo em função da dualidade desempenho máximo *versus* facilidade de uso.

Começando na arquitetura de máquinas paralelas, passando pelas bibliotecas de suporte e linguagens de programação para finalmente atingir os padrões de paralelismo, acreditamos que esses conceitos podem e devem ser abordados na forma de camadas com o objetivo de abstrair o nível inferior e oferecer um conjunto de serviços relativo ao conceito tratado.