ISPC:A SPMP Compiler for High-Performance CPU Programming

O paralelismo SIMD (Single Instruction Multiple Data) tem se tornado num mecanismo importante para melhorar o rendimento em CPU's modernas, devido a sua eficiência de energia e custo relativamente baixo na área comparado com outras formas de paralelismo.
Infelizmente, linguagems ou compiladores para os CPUs não tem essas capacidades.
Os modelos existentes de computação paralela se focan só no paralelismo multi-núcleo, negligenciando as capaciadades computacionais que estão disponíveis nas unidades vectoriales dos CPUs SIMD. Os linguagems orientados a GPU como o OpenCL,CUDA dão supporte a SIMD mas não tem a capacidade necessária para alcançar a máxima eficiência já que eles foram desenvolvidos para o GPU e não para o CPU.

Alcançar a uma alta eficiência utilizando o modelo SPMD é muito difícil na prática, embora que as técnicas para paralelizar através dos núcleos do computador são bem conhecidos e podem ser aprendidos facilmente, chegar a paralelizar através de operações vectoriais SIMD, se requere escrever códigos intrínsecos para gerar as sequências de códigos.

Os linguagens paralelos mais comunes desenhados para os CPUs incluindo OpenMP,MPI,Thread Building Blocks, UPC e Cilk se focam no paralelismo multi-núcleos e não fornecem o paralelismo SIMD no núcleo. Também se tentou utilizar o OpenCL para alcançar o paralelismo SIMD mas não se conseguiu já que o OpenCL não tem as capacidades necessárias para obter a máxima eficiência em CPUs e impõe penas de produtividade causadadas pela necessidade de acomodar as limitações do GPU como o sistema de memória separada. Já que se apresentarom esse tipod e problemas para desenvolveram o ISPC para obter a máxima eficiência e productividade usando as unidades SIMD dos CPUs modernos.

O compilador ISPC (Intel SPMD Program Compiler), é uma extensão do linguagem C para executar SPMD utilizando o paralelismo SIMD; o modelo de execução do ISPC tem como principal característica a execução de programas com fluxos de controle divergentes entre as pistas SIMD. O foco principal do ISPC é o uso eficiênte das unidades SIMD do CPU, também suporta o paralelismo multi-núcleo.
O linguagem e o modelo de programação subjacente foram desenhados para obter a máxima capacidade do CPU moderno, proporcionando a facilidade de uso e alta produtividade do programador. A escala do desempenho dos programas escritos em ISPC geralmente são obtidos do produto do número de núcleos e a largura de suas unidades SIMD. As características mais importantes do ISCP para o desempho são:

1. Suporte explícito do linguagem para as operações escalares e SIMD.
2. Supporte para estructuras de dados com estrutura de arranjos.
3. Acesso ao hardware subjacente do CPU, incluindo a capacidade de lançar tarefas

assíncronas e para executar operações rápidas entre unidades SIMD.

As características mais importantes do ICPC para seu uso são:

1. suporte para o acoplamento forte entre o C++ e ISPC, incluindo a capacidade de chamar diretamente rotinas de C++ desde ISPC ou ao contrário.
2. Memoria compartilhada coerente entre C++ e ISPC.
3. Sintaxe e características familiares ao C.

O alvo do ISPC são os programadores focados no desempenho. Portanto, um das principais vantagens do ICPC e que provê transparência no desempenho: como o C, é fácil para o usuário entender como o código é compilado ao hardware e como o código vai se executar. Um usuário habilidoso poderia obter um desempenho similar a aqueles que são alcançados códigos programados com funções intrínsecas implementados em SSE e AVX.
Outra vantagem é que a produtividade do programador que implementa seus códigos em ISPC é maior do que programar com funções intrínsecas, e é comparável com aqueles que escrevem código serial em C de alto desempenho ou os kernels OpenCL. Outra vantagem do ISCP é sua facilidade de adoção e interoperabilidade, já que é uma extensão do C e este linguagem pode ser utilizado com C/C++.

O ISPC não tem suporte para GPU, já que as arquiteturas do CPU e GPU são diferentes, além disso o ISPC não provê paralelismo "seguro" já que não protegem ao programador de cair num deadlock ou uma condição de carreira.

Já que o linguagem ISPC foi desenhado para prover alta eficiência em CPUs modernos, é útil rever algumas das características desse hardware:

Paralelismo Multinúcleo e SIMD: Um CPU moderno consiste de muitos núcleos, cada um deles com uma unidade escalar e uma unidade SIMD. As instruções para acessar à unidade SIMD têm nomes diferentes em diferentes arquiteturas: SSE para uma unidade SIMD com largura de 128 bits em processadores x86, AltiVec em processadores PowerPC e Neon em processadores ARM. O ISPC em sua versão atual suporta SSE ,AVX e Neon

Execução simultánea de instruções SIMD: As arquiteturas dos CPUs modernos podem realizar multiples instruções por ciclo quando as unidades de execução apropriadas estão disponíveis para essas instruções.

Un contador de programa por núcleo: A unidade escalar e todas las pistas das unidades associadas SIMD compartilham um único contador de programa.

Uma única memória coerente: Todos os núcleos compartilham um único endereço coerente com o cachê e memória do sistema para as operações escalares e as operações SIMD. Esta característica simplifica muito o compartilhamento de

estrutura de dados entre o código serial e paralelo. Esta é a característica que não tem os GPUs
A ordem de execução e memória estão bem definidas: Os computadores modernos tem réguas relativamente estritas nos quais as instruções tem que ser executadas e as réguas para salvar na memória , os GPUs tem réguas mais relaxadas, o qual permite grão flexibilidade para que o agendamento do hardware mas faz que seja mais difícil prover garantias de ordem no nível do linguagem.

Todos os linguagens paralelos precisam um modelo conceitual para expressar o paralelismo no linguagem e fazer o mapeamento de este paralelismo do linguagem no hardware. Um dos objetivos do grupo que desenvolveu o ISPC foi desenvolver um linguagem e um compilador para computadores modernos. Umas das opções era de criar um linguagem puramente sequencial, modificação do C, e deixar ao compilador achar as regiões paralelizáveis e mapeá-lo no hardware. Esse método é conhecido como “auto-vectorization”. Embora que a “auto-vetorização” trabalhe bem para código regular que não utiliza controles condicionais, limita a aplicabilidade dessa técnica na prática. Todas as otimizações que são feitas pelo “auto-vectorizator” tem que respeitar a semântica sequencial do código; portanto o auto vetorizador deve ter a visibilidade de todo o corpo do loop, o qual opõe que dentro do loop sejam chamados outras funções, por exemplo. Uma função com controle de fluxo complexo e chamadas de funções aninhadas inibe a auto-vetorização na prática. Portanto é dificial saber se uma região de código vai se paralelizar ou não. Tendo em conta todos esses problemas os desenvolvedores do ISPC escolheram fazer a paralelização do código utilizando as intruções SIMD do computador, mas fazer um programa utilizando só código intrinseco é difícil de fazer.

Dado que fazer a vetorização de um programa utilizando as instruções SIMD é difícil, portanto desenvolveram o ISPC utilizando a abstração SPMD onde o programa cria várias instâncias onde cada um deles executa o mesmo código só que com diferentes dados.
O conjunto de instâncias que se executam no ISPC é chamado de “gang” (o ISPC garante que as instâncias que se estão executando no “gang” são executados todos ao mesmo tempo). O número de instâncias do programa num “gang” é relativamente pequeno; na prática, não há mas de duas vezes a largura da unidade vetorial do computador no qual se está executando. Assim, há quatro, oito ou até dezesseis instâncias num gang num CPU utilizando o conjunto de instruções SSE de largura 4, AVX de largura 8 ou Xeon Phi de largura 16. O tamanho do “gang” é determinado em tempo de compilação, e tem que ser especificádo pelo usuário.

Uma dos retos do modelo de execução SPMD é o controle de fluxos de controle divergentes. Consideremos um loop “while” com um teste de termino “n > 0”; quando diferentes instâncias do programa tem diferentes valores de “n”, eles precisam de executar o loop um número diferente de vezes.

O modelo SPMD-SIMD do ISPC provê a ilusão de um fluxo de controle separado para cada uma das pistas do SIMD, mas essa ilusão é criada no compilador o qual utiliza mascaras para saber qual pista está ativa, e depois executa o corpo do loop só para aquelas pistas que estejam ativas. A figura 1 mostra um exemplo do modelo de execução.

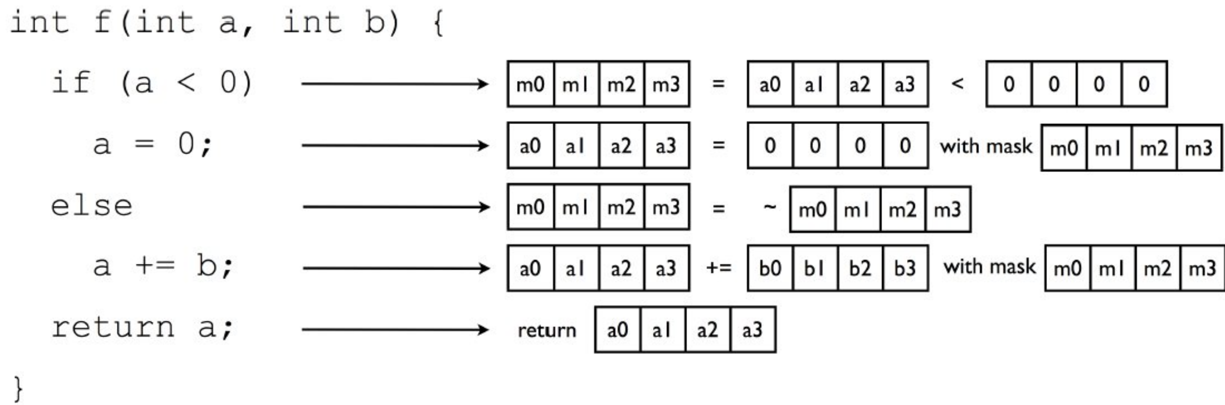

Illustration 1: Exemplo da utilização de mascaras para o fluxo de controle no ISPC.[1]

Para mostrar um pouco da sintaxe e como o linguagem e utilizado, se mostrará um exemplo do ISPC. Para um melhor entendimento e maior quantidade de exemplos pode se ver as páginas [2] e [3].

Primeiro, temos um código em C++ que de maneira dinâmica aloca e inicializa dois arranjos. Depois chama a função "update".

```
float *values = new float[1024];
int *iterations = new int[1024];
// ... initialize values[], iterations[] ...
update(values, iterations, 1024);
```

Illustration 2: main.c [1]

A chamada a "update()" é uma chamada regular de uma função; mas em este caso o "update" foi implementado em ISPC. Essa função eleva ao quadrado cada elemento do arranjo "value" o número de vezes indicado no arranjo "iterations".

```
export void update(uniform float values[],
                   uniform int iterations[],
                   uniform int num_values) {
    for (int i = programIndex; i < num_values;
         i += programCount) {
        int iters = iterations[i];
        while (iters-- > 0)
            values[i] *= values[i];
    }
}
```

Illustration 3: update.ispc[1]

A sintaxe e as capacidades do ISPC estão baseados em C/C++, como se pode observar o código é bem parecido a um código C/C++ o qual faz que seja mais fácil aprender o linguagem.
A função “update” tem um qualificador “export”, o qual indica que essa função pode ser chamada desdo o C++; a qualificador de variável “uniform” especifica que debe ser salvado e executado como um dado escalar.
Dado um número de instâncias do programa executando-se em SPMD, é necessário para as instâncias iterar sobre os dados de entrada (tipicamente maior que o tamanho de uma “gang”). No exemplo que foi dado cada instância itera sobre os dados utilizando o loop “for” e as variáveis construídas “programIndex” e “ProgramCount”. “programCount” devolve o número total de instâncias que se estão executando (o tamanho do “gang”) e “programIndex” é o identificador da instância que tem valores de zero até “programCount – 1”. Assim, no código ISPC para cada iteração do loop “for” um conjunto de elementos de tamanho “programCount” dos dados de entrada são processados paralelamente pelas instâncias do programa.
A variável “programIndex” é análogo ao “threadIdx” do CUDA e “get_global_id()” di OpenCL.

Num linguagem SPMD como o ISPC, a declaração de uma variável como “float x” representa uma varíavel com uma localização separada de memória (podem ter valores diferentes) para cada uma das instâncias do programa. Entretanto, algumas variáveis e seus cálculos associados não precisam ser feitos em todas as instâncias do programa. Por exemplo, o cálculo de endereços e variáveis de iteração de loops podem ser compartilhadas.
Como o CPU provê unidades de operação escalar e vetorial separadas, é importante não fazer operações nem endereços de memória replicados.
O ISPC provê uma classe de salvamento “uniform” para este propósito, o qual corresponde a um único valor na memória e assim, um valor que é igual para todos os elementos.

Até agora os exemplos mostrados são executados num processador só, portanto o ISPC provê um mecanismo de lançamento de tarefas assíncrono. Se define uma função do ISPC como “task” são semanticamente chamadas de funções assíncronas que podem ser executados concorrentemente em diferentes cores. Esta capacidade faz que a paralelização dos programas ISPC direto, geralmente só é necessário poucas linhas de código para executar em várias linhas de código, como mostrado na figura 4.

```
const uniform int task_size = 4096;

task void sum_task(uniform int a[], uniform int count, uniform int &sum){
        uniform int start = task_size*taskIndex;
        uniform int end = min(task_size*(taskIndex + 1),count);
        int partial = 0;
        foreach(i = start … end) partial += a[i];
        uniform int local_sum = reduce_add(partial);
        atomic_add_global(&sum,local_sum);
}

export array_sum(uniform int a[],uniform int count){
        int n_tasks = count/task_size;
        uniform int sum = 0;
        launch [n_task] sum_tasks(a,count,sum);
        return sum;
}
```

Illustration 4: código ISPC

O algoritmo “array_sum” retorna a soma de todos os elementos do arranjo “a” chamando a função “sum_tasks” do qual se criam “n_task” instâncias, conde cada instância calcula a soma dos elementos de um pedaço do arranjo, utiliza uma operação reduze para calcular a coma total (já que cada pista tem um valor), e no final se calcula a soma de todas as tarefas utilizando “atomic_add_global”.

Umas das vantagens do SIMD na CPU e o conjunto rico de operações inter-pistas. Por exemplo, existem operações para mandar o valor de uma pista a todas as outras, intrusões para permutar os valores das pistas. O ISPC expõe essas capacidades através de funções que permitem as instâncias num “gang” intercambiar dados.

REFERÊNCIAS

[1] ispc: A SPMD Compiler for High-Performance CPU Programming. **Matt Pharr e William R. Mark.**
[2] http://pharr.org/matt/ Criador do ispc.
[3] https://ispc.github.io/ Código fonte, documentação e exemplos do ISPC.