Нижегородский государственный университет им. Н.И.Лобачевского

Семинар по высокопроизводительным вычислениям лаборатории ИтЛаб

Технология Ct

Сиднев А.А.
Кафедра математического обеспечения ЭВМ, факультет ВМК

Предварительный вариант

Развитие современных процессоров

- Усложнение архитектуры параллельных систем приводит к усложнению разработки параллельных программ под эти архитектуры:
  - необходимо упростить разработку;
  - необходимо разрабатывать эффективные программы для систем с различным количеством вычислительных ядер/процессоров.

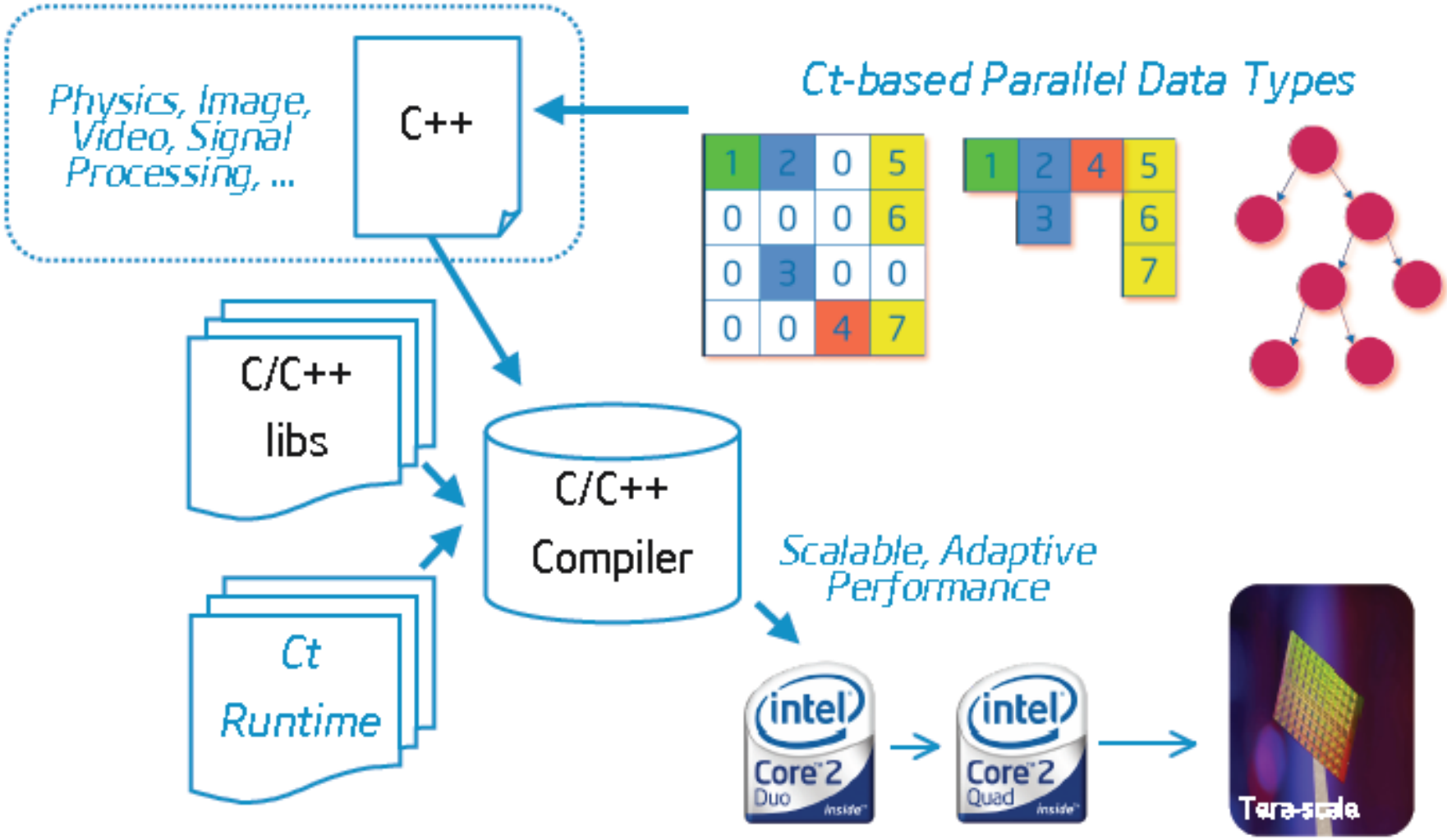
Технологии создания параллельных программ

- MPI (Message Passing Interface)
- WinThreads (pthreads)
- Fork-join
  - OpenMP
- Task Based
  - TBB (Threading Building Blocks)
- Streaming (flat) data parallelism
  - CUDA
  - OpenCL
- Nested data parallelism
  - Ct

Data parallelism

- Flat data parallel models:
  - ограниченная размерность векторов;
  - ограниченное количество коллективных операций (reduction);
  - примеры: APL, F90/HPF, GPGPU.
- Nested data parallel models:
  - поддерживаются вложенные/индексные вектора;
  - большое количество коллективных операций;
  - примеры: Nesl, APL2, Paralations.

Использование Ct



Ct

- Основу Ct составляет базовый тип TVEC:
  - управляется Ct runtime;
  - может быть:
    - плотный,
    - многомерный,
    - разреженный,
    - вложенный;
  - значения могут быть созданы и изменены только с помощью Ct API.

Представление матриц

- Плотные матрицы

```
for (row = 0; row < row_num; row++) {
   for (col = 0; col < col_num; col++) {
       //Используем A[row][col]
   }
}
```

- Разреженные матрицы

Разреженные матрицы

- CSC (Compressed Sparse Column) формат:
  - В данном формате вместо одного двумерного массива, используются три одномерных.
  - Значения ненулевых элементов матрицы и соответствующие им сроковые индексы хранятся в этом формате по строкам в двух массивах `Values` и `RowIdx`.
  - Массив указателей `ColP`, используется для ссылки на компоненты массивов `Values` и `RowIdx`, с которых начинается описание очередного столбца.

Разреженные матрицы (пример)

- Матрица:

```
A = [ [0 1 0 0 0]
      [2 0 3 4 0]
      [0 5 0 0 6]
      [0 7 0 0 8]
      [0 0 9 0 0] ]
```

- CSR (Compressed Sparse Row) формат:

```
Values = [2 1 5 7 3 9 4 6 8]
RowIdx = [1 0 2 3 1 4 1 2 3]
ColP   = [0 1 4 6 8 9]
```

Матрично-векторное умножение

```
for (c = 0; c < col_num; c++) {
   for (e = ColP[c]; e < ColP[c + 1]; e++){
       int r = RowIdx[e];
       product[r] += Values[e] /* A[r][c] */ * v[c];
   }
}
```

- Количество итераций внутреннего цикла может быть различным, поэтому сбалансировать нагрузки всех вычислительных потоков будет тяжело (при классическом распараллеливании внешнего цикла)
- При обработке элемента матрицы могут возникнуть зависимости (в более сложных алгоритмах)

Матрично-векторное умножение с помощью CUDA

- Около 120 строк

Listing 4. The proposed CUDA program for matrix-vector multiplication

```
// y = Ax
// A : m-by-n matrix, x : n elements vector, y : m elements vector
// m and n are arbitrary positive integers.

texture<float4, 2, cudaReadModeElementType> texRefA;

void mv(float *y, float *A, float *x, int m, int n) {
    int blkNum = (m >> 6) + ((m & 63) ? 1 : 0);
    int height = blkNum << 6;
    int width = (n & 511) ? (((n >> 9) + 1) << 9) : n;
    dim3 threads(8, 64, 1), grid(blkNum, 1);
    cudaArray *d_A; float *d_x, *d_y;

    cudaChannelFormatDesc channelDesc=cudaCreateChannelDesc<float4>();
    cudaMallocArray(&d_A, &channelDesc, width >> 2, height);
    cudaMemcpy2DToArray(d_A, 0, 0, A, n * sizeof(float),
                        n * sizeof(float), m, cudaMemcpyHostToDevice);
    cudaBindTextureToArray(texRefA, d_A);
    cudaMalloc((void**) &d_x, n * sizeof(float));
    cudaMalloc((void**) &d_y, m * sizeof(float));

    cudaMemcpy(d_x, x, n * sizeof(float), cudaMemcpyHostToDevice);
    mv_kernel<<< grid, threads >>>(d_y, d_A, d_x, m, n);
    cudaMemcpy(y, d_y, m * sizeof(float), cudaMemcpyDeviceToHost);

    cudaFree(d_y); cudaFree(d_x); cudaUnbindTexture(texRefA);
    cudaFreeArray(d_A);
}

#define bx blockIdx.x
#define tx threadIdx.x
#define ty threadIdx.y
__global__ void mv_kernel(float* y, cudaArray* A, float* x,int m,int n)
{
    __shared__ float xs[64][8]; __shared__ float Ps[64][8];
    float4 a; float *Psptr = (float *) Ps + (ty << 3) + tx;
    int ay = (bx << 6) + ty; float *xptr = x + (ty << 3) + tx;
    float *xsptr = (float *) xs + (tx << 2);

    *Psptr = 0.0f;
    int i;
    for (i = 0; i < (n & ~511); i += 512, xptr += 512) {
        xs[ty][tx] = *xptr;
        __syncthreads();
        int ax = tx + (i >> 2);
        a = tex2D(texRefA, ax       , ay);
        *Psptr += a.x * *xsptr          + a.y * *(xsptr +   1)
                  + a.z * *(xsptr +   2) + a.w * *(xsptr +   3);
        a = tex2D(texRefA, ax +   8, ay);
        *Psptr += a.x * *(xsptr +  32) + a.y * *(xsptr +  33)
                  + a.z * *(xsptr +  34) + a.w * *(xsptr +  35);
        a = tex2D(texRefA, ax +  16, ay);
        *Psptr += a.x * *(xsptr +  64) + a.y * *(xsptr +  65)
                  + a.z * *(xsptr +  66) + a.w * *(xsptr +  67);

// Continued to the next page
```

The proposed CUDA program for matrix-vector multiplication (Cont'd)

```
        a = tex2D(texRefA, ax +  24, ay);
        *Psptr += a.x * *(xsptr +  96) + a.y * *(xsptr +  97)
                  + a.z * *(xsptr +  98) + a.w * *(xsptr +  99);
        a = tex2D(texRefA, ax +  32, ay);
        *Psptr += a.x * *(xsptr + 128) + a.y * *(xsptr + 129)
                  + a.z * *(xsptr + 130) + a.w * *(xsptr + 131);
        a = tex2D(texRefA, ax +  40, ay);
        *Psptr += a.x * *(xsptr + 160) + a.y * *(xsptr + 161)
                  + a.z * *(xsptr + 162) + a.w * *(xsptr + 163);
        a = tex2D(texRefA, ax +  48, ay);
        *Psptr += a.x * *(xsptr + 192) + a.y * *(xsptr + 193)
                  + a.z * *(xsptr + 194) + a.w * *(xsptr + 195);
        a = tex2D(texRefA, ax +  56, ay);
        *Psptr += a.x * *(xsptr + 224) + a.y * *(xsptr + 225)
                  + a.z * *(xsptr + 226) + a.w * *(xsptr + 227);
        a = tex2D(texRefA, ax +  64, ay);
        *Psptr += a.x * *(xsptr + 256) + a.y * *(xsptr + 257)
                  + a.z * *(xsptr + 258) + a.w * *(xsptr + 259);
        a = tex2D(texRefA, ax +  72, ay);
        *Psptr += a.x * *(xsptr + 288) + a.y * *(xsptr + 289)
                  + a.z * *(xsptr + 290) + a.w * *(xsptr + 291);
        a = tex2D(texRefA, ax +  80, ay);
        *Psptr += a.x * *(xsptr + 320) + a.y * *(xsptr + 321)
                  + a.z * *(xsptr + 322) + a.w * *(xsptr + 323);
        a = tex2D(texRefA, ax +  88, ay);
        *Psptr += a.x * *(xsptr + 352) + a.y * *(xsptr + 353)
                  + a.z * *(xsptr + 354) + a.w * *(xsptr + 355);
        a = tex2D(texRefA, ax +  96, ay);
        *Psptr += a.x * *(xsptr + 384) + a.y * *(xsptr + 385)
                  + a.z * *(xsptr + 386) + a.w * *(xsptr + 387);
        a = tex2D(texRefA, ax + 104, ay);
        *Psptr += a.x * *(xsptr + 416) + a.y * *(xsptr + 417)
                  + a.z * *(xsptr + 418) + a.w * *(xsptr + 419);
        a = tex2D(texRefA, ax + 112, ay);
        *Psptr += a.x * *(xsptr + 448) + a.y * *(xsptr + 449)
                  + a.z * *(xsptr + 450) + a.w * *(xsptr + 451);
        a = tex2D(texRefA, ax + 120, ay);
        *Psptr += a.x * *(xsptr + 480) + a.y * *(xsptr + 481)
                  + a.z * *(xsptr + 482) + a.w * *(xsptr + 483);
        __syncthreads();
    }

    if (i + (ty << 3) + tx < n) {
        xs[ty][tx] = *xptr;
    }
    __syncthreads();
    int j;
    for (j = 0; j < ((n - i) >> 5); j++, xsptr += 29) {
        a = tex2D(texRefA, tx + (i >> 2) + (j << 3), ay);
        *Psptr += a.x * *xsptr++ + a.y * *xsptr++
                  + a.z * *xsptr++ + a.w * *xsptr;
    }
    __syncthreads();
    int remain = (n - i) & 31;
    if ((tx << 2) < remain) {
        a = tex2D(texRefA, tx + (i >> 2) + (j << 3), ay);
        *Psptr += a.x * *xsptr++;
    }
    if ((tx << 2) + 1 < remain) *Psptr += a.y * *xsptr++;
    if ((tx << 2) + 2 < remain) *Psptr += a.z * *xsptr++;
    if ((tx << 2) + 3 < remain) *Psptr += a.w * *xsptr;
    __syncthreads();

    if (tx < 4) *Psptr += *(Psptr + 4);
    if (tx < 2) *Psptr += *(Psptr + 2);
    if (tx < 1) *Psptr += *(Psptr + 1);

    __syncthreads();
    if (ty < 8)
        if ((bx << 6) + tx + (ty << 3) < m)
            y[(bx << 6) + tx + (ty << 3)] = Ps[tx + (ty << 3)][0];
}
```

Матрично-векторное умножение с помощью Ct

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values,
TVEC<I32> RowIdx,
TVEC<I32> ColP,
TVEC<F64> v)
{
   TVEC<F64> expv = distribute(v,ColP);
   TVEC<F64> product = Values*expv;
   product = product.applyNesting(RowIdx, ctIndex);

   TVEC<F64> result = product.addReduce();
   return result;
}
```

Матрично-векторное умножение (1)

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values, // [2 1 5 7 3 9 4 6 8]
TVEC<I32> RowIdx, // [1 0 2 3 1 4 1 2 3]
TVEC<I32> ColP,   // [0 1 4 6 8 9]
TVEC<F64> v)      // [1 2 3 4 5]
{
   TVEC<F64> expv = distribute(v,ColP);
   TVEC<F64> product = Values*expv;
   product = product.applyNesting(RowIdx, ctIndex);

   TVEC<F64> result = product.addReduce();
   return result;
}
```

Матрично-векторное умножение (2)

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values, // [2 1 5 7 3 9 4 6 8]
TVEC<I32> RowIdx, // [1 0 2 3 1 4 1 2 3]
TVEC<I32> ColP,   // [0 1 4 6 8 9]
TVEC<F64> v)      // [1 2 3 4 5]
{
   TVEC<F64> expv = distribute(v,ColP); // [1 2 2 2 3 3 4 5 5]
   TVEC<F64> product = Values*expv;
   product = product.applyNesting(RowIdx, ctIndex);

   TVEC<F64> result = product.addReduce();
   return result;
}
```

Матрично-векторное умножение (3)

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values, // [2 1 5 7 3 9 4 6 8]
TVEC<I32> RowIdx, // [1 0 2 3 1 4 1 2 3]
TVEC<I32> ColP,   // [0 1 4 6 8 9]
TVEC<F64> v)      // [1 2 3 4 5]
{
   TVEC<F64> expv = distribute(v,ColP); // [1 2 2 2 3 3 4 5 5]
   TVEC<F64> product = Values*expv; // [2 2 10 14 9 27 16 30 40]
   product = product.applyNesting(RowIdx, ctIndex);

   TVEC<F64> result = product.addReduce();
   return result;
}
```

Матрично-векторное умножение (4)

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values, // [2 1 5 7 3 9 4 6 8]
TVEC<I32> RowIdx, // [1 0 2 3 1 4 1 2 3]
TVEC<I32> ColP,   // [0 1 4 6 8 9]
TVEC<F64> v)      // [1 2 3 4 5]
{
   TVEC<F64> expv = distribute(v,ColP); // [1 2 2 2 3 3 4 5 5]
   TVEC<F64> product = Values*expv; // [2 2 10 14 9 27 16 30 40]
   product = product.applyNesting(RowIdx, ctIndex);
  // [[2] [2 9 16] [10 30] [14 40] [27]]
   TVEC<F64> result = product.addReduce();
   return result;
}
```

Матрично-векторное умножение (5)

```
TVEC<F64> sparseMatrixVectorProducSC(
TVEC<F64> Values, // [2 1 5 7 3 9 4 6 8]
TVEC<I32> RowIdx, // [1 0 2 3 1 4 1 2 3]
TVEC<I32> ColP,   // [0 1 4 6 8 9]
TVEC<F64> v)      // [1 2 3 4 5]
{
   TVEC<F64> expv = distribute(v,ColP); // [1 2 2 2 3 3 4 5 5]
   TVEC<F64> product = Values*expv; // [2 2 10 14 9 27 16 30 40]
   product = product.applyNesting(RowIdx, ctIndex);
  // [[2] [2 9 16] [10 30] [14 40] [27]]
   TVEC<F64> result = product.addReduce(); // [2 27 40 54 27]
   return result;
}
```

Обработка изображений (1)

- Преобразование цвета изображения

```
TVEC<F32> colorConvert(TVEC<F32> rchannel, TVEC<F32>
  gchannel, TVEC<F32> bchannel, TVEC<F32> achannel,
F32 a0, F32 a1, F32 a2, F32 a3)
{
return (rchannel * a0 + gchannel * a1 + bchannel * a2 +
  achannel * a3);
}
```

Обработка изображений (2)

- Фильтрация изображений

```
TVEC<F32> Convolve2D3x3(TVEC<F32> pixels, I32 channels,
   TVEC<F32> kernel) {
   TVEC<F32> respixels;
   respixels += shiftPermute(pixels, directions[0][0]) *
   kernel[0][0];
   respixels += shiftPermute(pixels, directions[0][1]) *
   kernel[0][1];
   //…
   return respixels
}
```

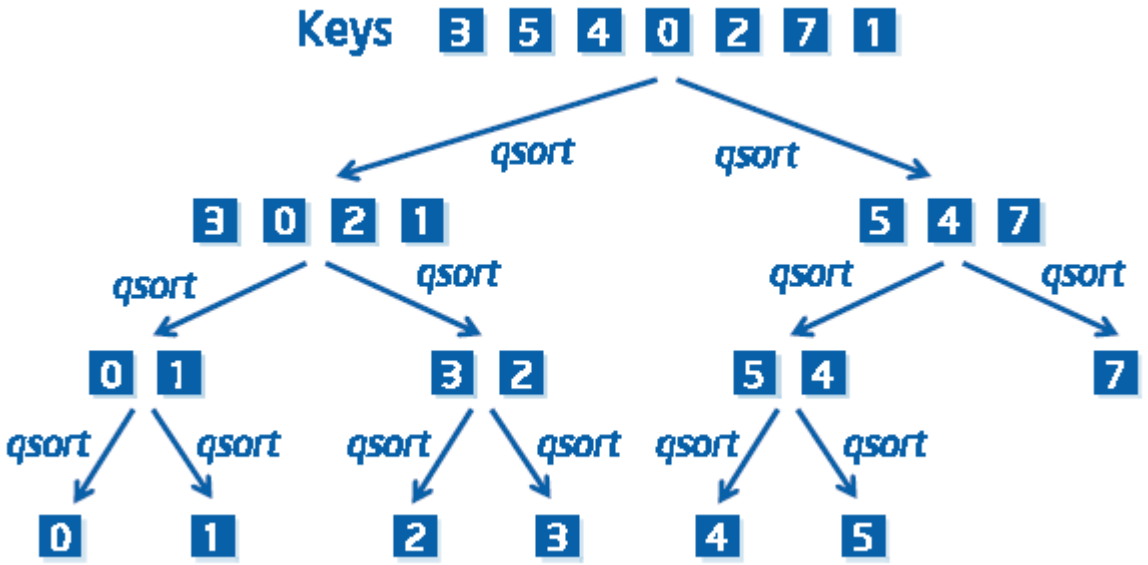
Обработка изображений (2)

- Фильтрация изображений (полный листинг)

```
TVEC<F32> Convolve2D3x3(TVEC<F32> pixels, I32 channels, TVEC<F32>
   kernel) {
   TVEC<F32> respixels;
// directions[m][n] is a constant TVEC of size 2 with values {m-1, n-1}
   respixels += shiftPermute(pixels, directions[0][0]) * kernel[0][0];
   respixels += shiftPermute(pixels, directions[0][1]) * kernel[0][1];
   respixels += shiftPermute(pixels, directions[0][2]) * kernel[0][2];
   respixels += shiftPermute(pixels, directions[1][0]) * kernel[1][0];
   respixels += pixels * kernel[1][1];
   respixels += shiftPermute(pixels, directions[1][2]) * kernel[1][2];
   respixels += shiftPermute(pixels, directions[2][0]) * kernel[2][0];
   respixels += shiftPermute(pixels, directions[2][1]) * kernel[2][1];
   respixels += shiftPermute(pixels, directions[2][2]) * kernel[2][2];
   return respixels
}
```

Сортировка

- Идея сортировки



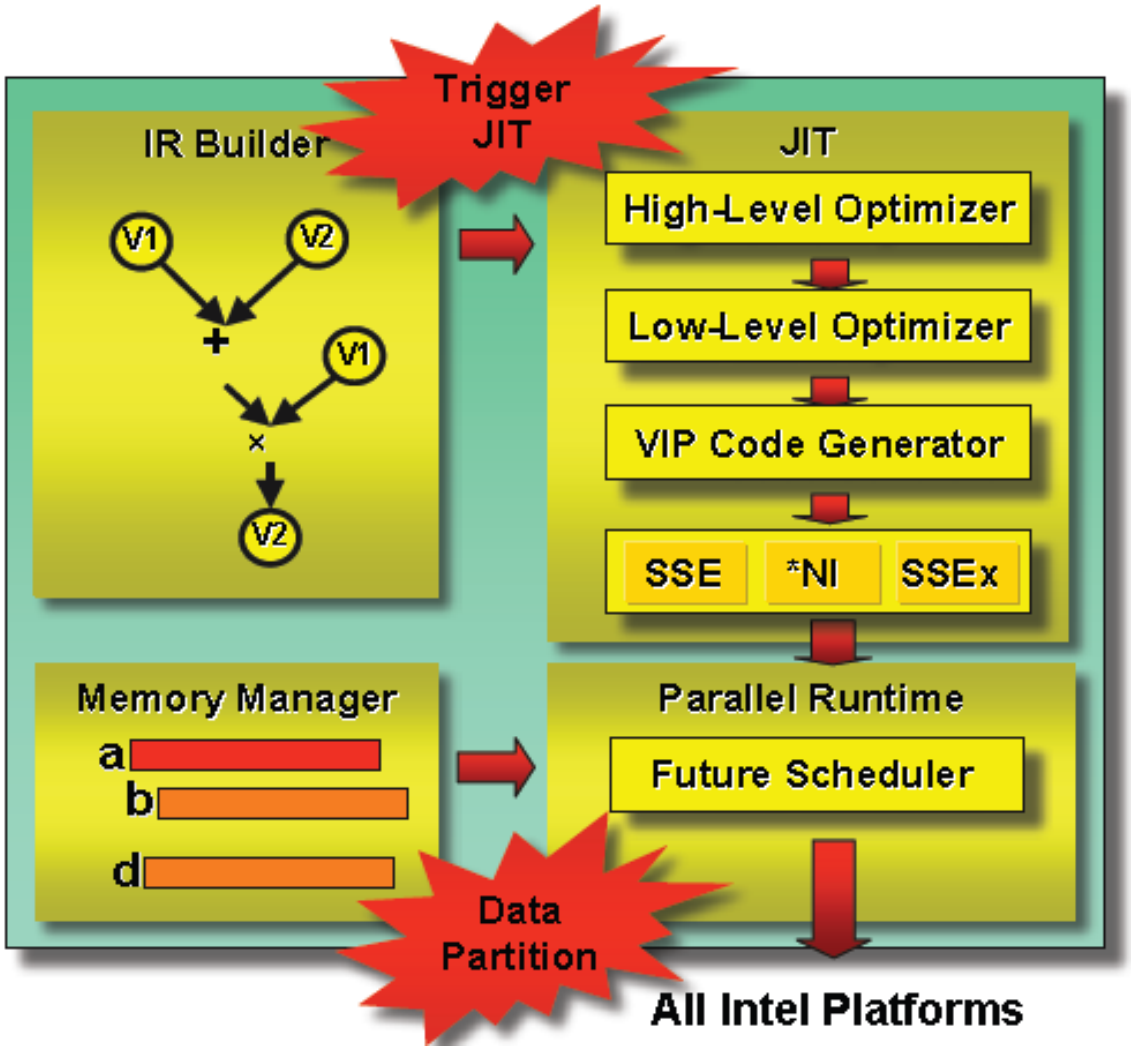
Сортировка

```
TVEC<F64> ctQsort(TVEC<F64> Keys) {
   TVEC<F64> pivot, lowerKeys, pivotKeys, upperKeys;
   TVEC<Bool> pivotFlags;
   I32 pivot;
   if (length(Keys) == 0)
       return Keys;
   pivot = extract(Keys, 0);
   pivotFlags = lessThan(Keys, Pivot);
   lowerKeys = pack(Keys, pivotFlags);
   pivotFlags = equal(Keys, Pivot);
   pivotKeys = pack(Keys, pivotFlags);
   pivotFlags = greaterThan(Keys, Pivot);
   UpperKeys = pack(Keys, pivotFlags);
   return cat(ctQsort(lowerKeys),
   cat(pivotKeys,ctQsort(upperKeys));
```

Выполнение Ct

```
float src1[], src2[], dest[];

TVEC<F32> a(src1), b(src2);
TVEC<F32> c = a + b;
TVEC<F32> d = c * a;
d.copyOut(dest);
```

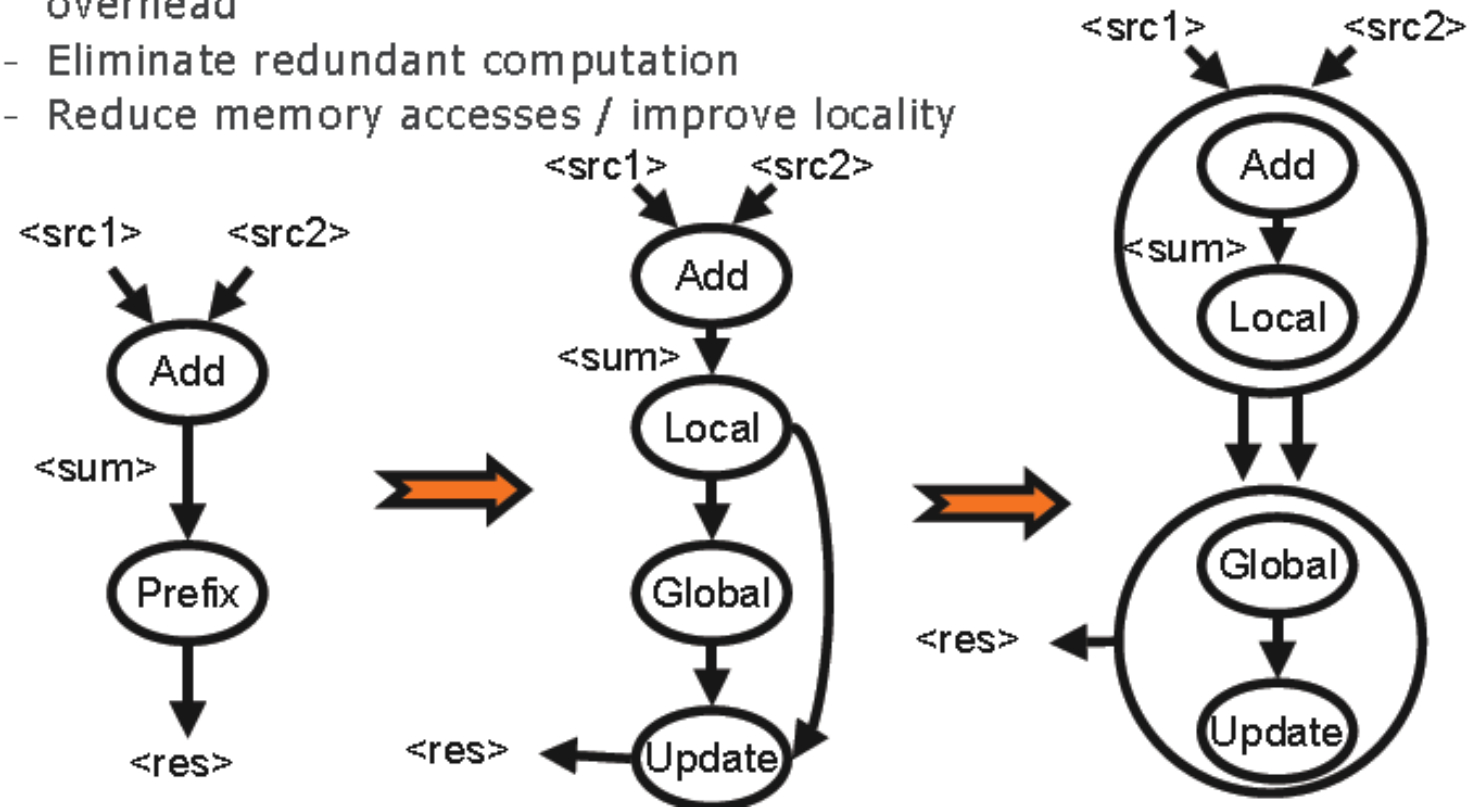


Ct Dynamic Engine

Высокоуровневая оптимизация

High-Level Optimizer

- ~20 optimizations (including classic opts)
- Increase granularity of parallelism / decrease threading overhead
- Eliminate redundant computation
- Reduce memory accesses / improve locality

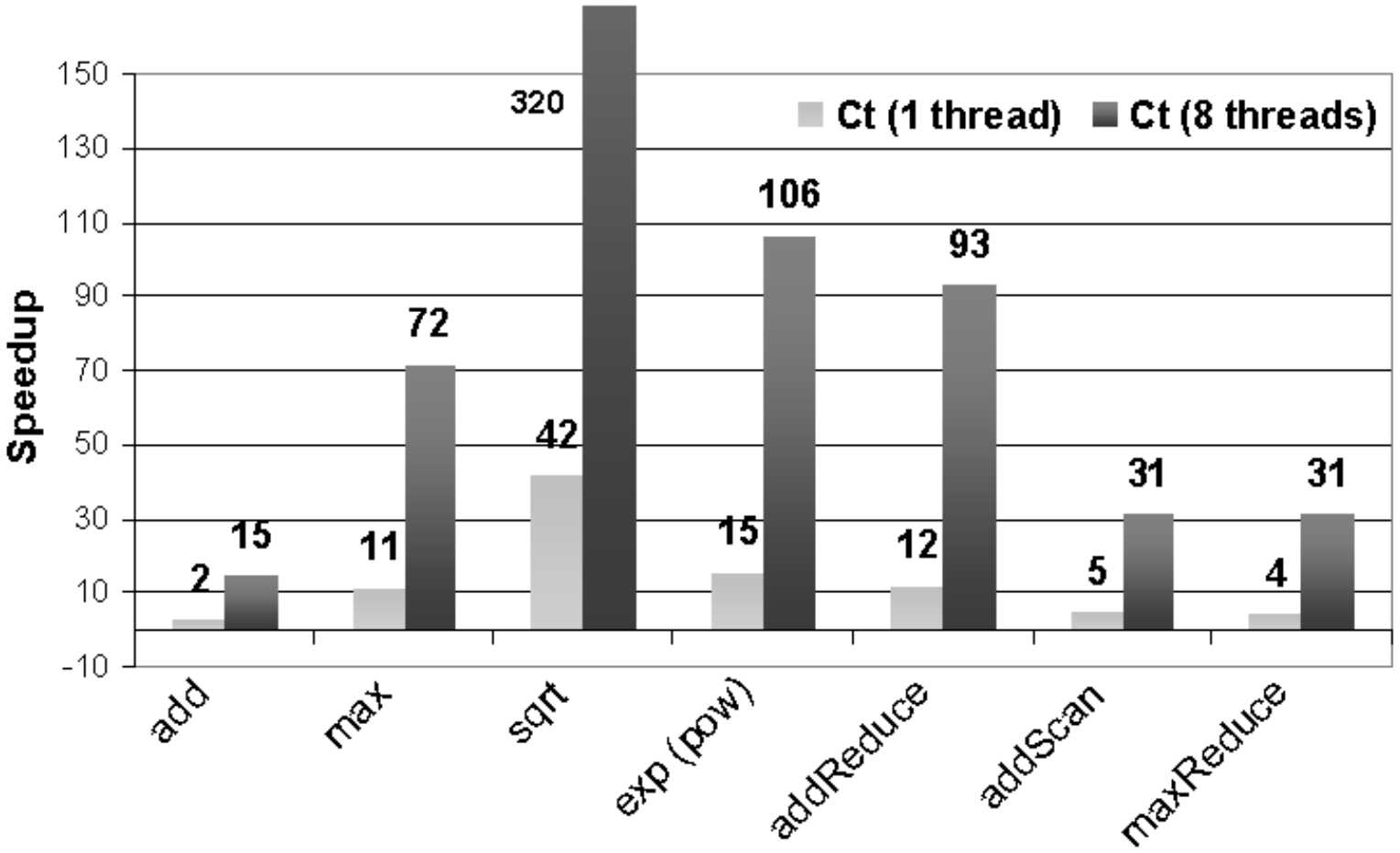


Низкоуровневая оптимизация

Low-Level Optimizer

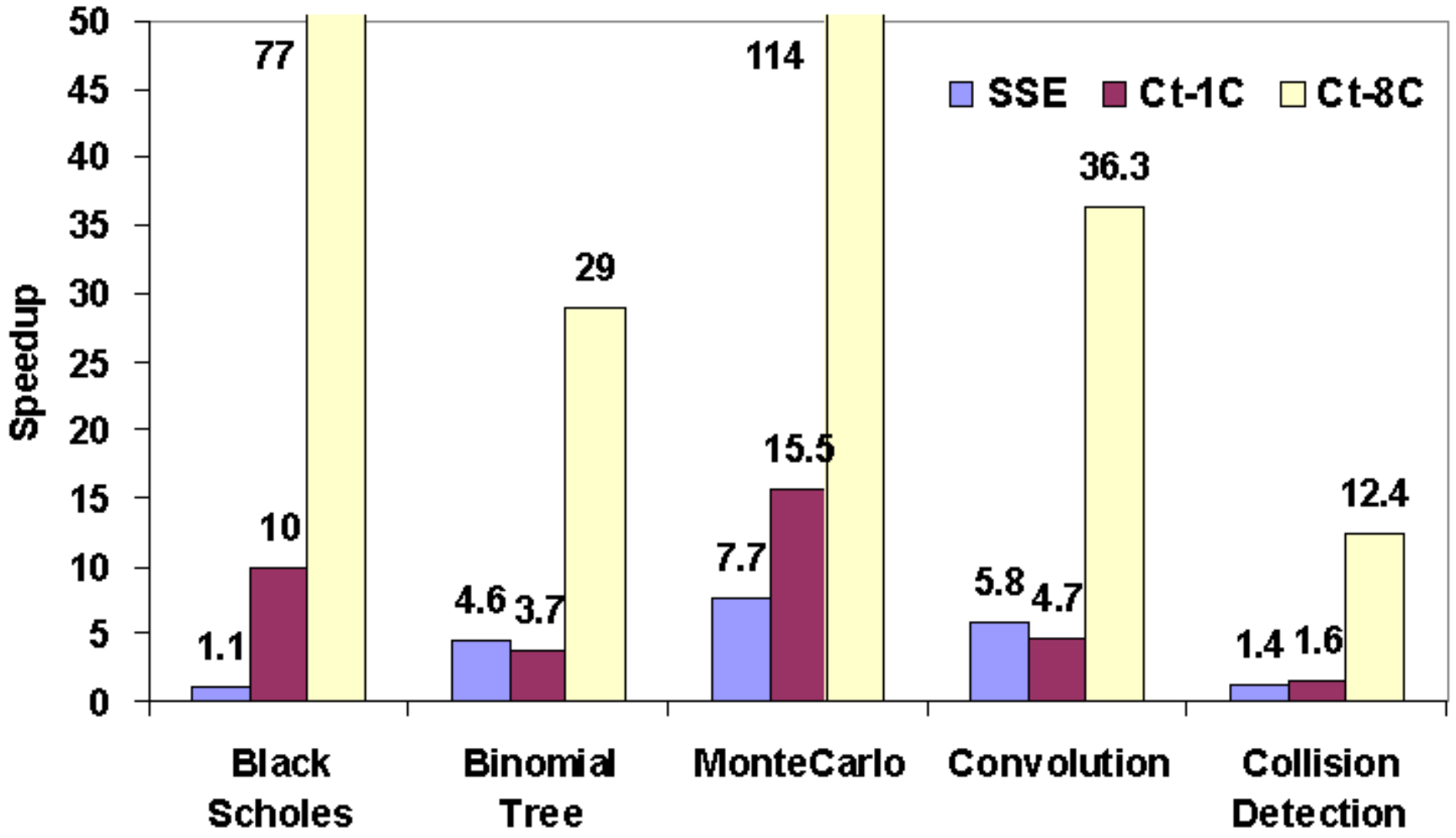
- ~10 optimizations
- Eliminate redundant checks.
- Reorganize the data layout.
- Parallelize the data-parallel tasks on multi threads.
- SIMD-vectorize each thread.

Эффективность операций



Intel® Xeon® processor E5345 platform (two 2.33GHz quad-core processors, 4GB memory)

Эффективность реализаций



Intel® Xeon® processor E5345 platform (two 2.33GHz quad-core processors, 4GB memory)

Объявления

```
TVEC<I8> Red;
Red = copyin(CRed, Height*Width, I8); // Red <- Cred

TVEC<F32> Xes;
copyout((void*)CXes,Xes); // CXes <- Xes
```

Операторы

TVEC<F32> A = B + C;

- Vector-Vector
  - add, sub, mul, div, equal, min, max, mod, lsh, rsh, greater, less…
- Vector-Scalar
  - addVectorScalar, subVectorScalar, subScalarVector, mulVectorScalar, divVectorScalar, divScalarVector, equalVectorScalar, minVectorScalar, maxVectorScalar…
- Unary
  - abs, not, log, exp, sqrt, rsqrt, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh…

Коллективные операции (1)

- Редукция

```
addReduce([1 0 2 -1 4]) // 6
```

- Префиксная сумма

```
addPrefix([1 0 2 -1 4]) // [0 1 1 3 2]
```

Коллективные операции (2)

- Редукция

```
addReduce([1 0 2 -1 4]) // 6
```

- Префиксная сумма

```
addPrefix([1 0 2 -1 4]) // [0 1 1 3 2]
```

- Reduction
  - addReduce, mulReduce, minReduce, maxReduce, andReduce, iorReduce, xorReduce, reduce
- Scan/Prefix-Sum
  - addScan, mulScan, minScan, maxScan, andScan, iorScan, xorScan, scan

Перемещение данных

- Pack/Unpack
  - pack, unpack
- Scatter/Gather
  - scatter, gather
- Shift/Rotate
  - leftShiftPermute, rightShiftPermute, leftRotatePermute, rightRotatePermute, shiftDefaultPermute, rotateDefaultPermute
- Partition
  - partition, unpartition
- Miscellaneous
  - defaultPermute, omegaPermute, butterflyPermute, distribute

Источники

- Ct: A Flexible Parallel Programming Model for Tera-scale Architectures. October 25, 2007.
- Future-Proof Data Parallel Algorithms and Software on Intel® Multi-Core Architecture. Intel® Technology Journal. Volume 11. November 15, 2007. ISSN 1535-864X.
- Anwar Ghuloum, Gansha Wu, Xin Zhou, Peng Guo, Jesse Fang. Programming Option Pricing Financial Models with Ct. October 17, 2007.
- Noriyuki Fujimoto. Dense matrix-vector multiplication on the CUDA architecture. Department of Mathematics and Information Sciences, Graduate School of Science, Osaka Prefecture University. June 20, 2008.

Вопросы?