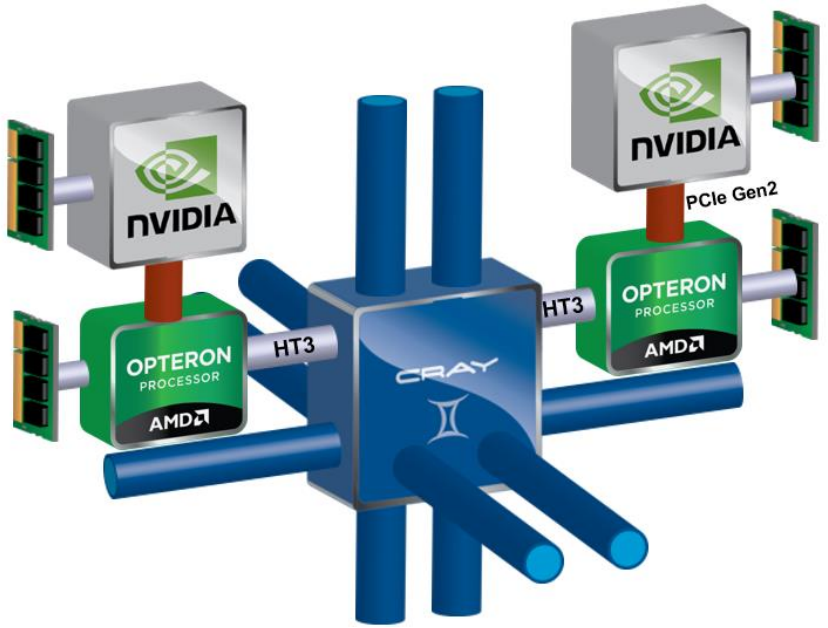
Два гибридных (гетерогенных) узла системы Cray XK7

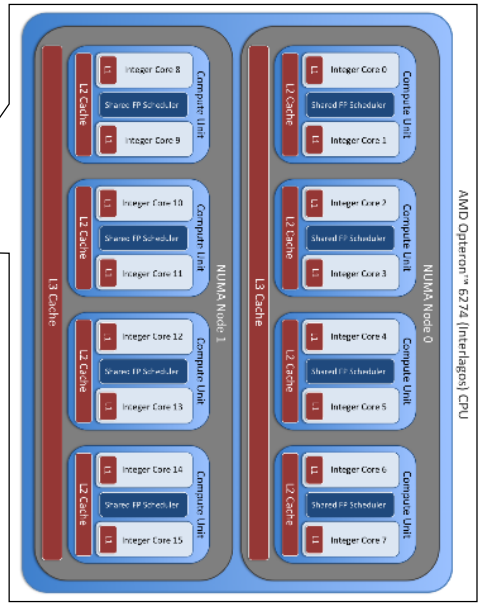
Уровень узла:

GPU: NVIDIA CUDA,
OpenCL,
OpenACC,
OpenMP 4.x

CPU: OpenMP, Cilk/Cilk++,
Intel TBB, POSIX Threads

Core: SSE/AVX





Процессор AMD Opteron
Interlagos (16 ядер)

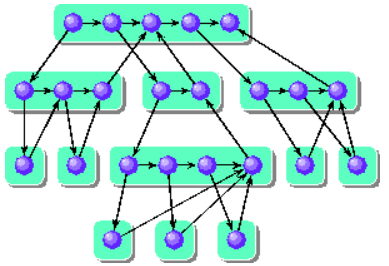
Уровень системы:

MPI, SHMEM,
Cray Chapel, Coarray Fortran, Unified Parallel C

Intel Cilk Plus

Язык программирования Cilk

- **Cilk** – расширение языка ANSI C для создания многопоточных программ
- **The Cilk Project:** разработка начата в 1994 г. в Лаборатории компьютерных наук Массачусетского института технологий (MIT, USA)
- Один из разработчиков и руководителей проекта – Charles E. Leiserson
- В 2006 г. C.E. Leiserson основал компанию **Cilk Arts** для продвижения на рынок языка Cilk
- В 2007 г. выходит Cilk++ (Cilk с поддержкой C++)



http://supertech.csail.mit.edu/cilk/

- В 2009 г. компания Intel приобрела Cilk Arts и выпустила **Intel Cilk Plus**
- Intel поддерживает Cilk Plus в Development-ветви GCC 4.8 и GCC 4.9

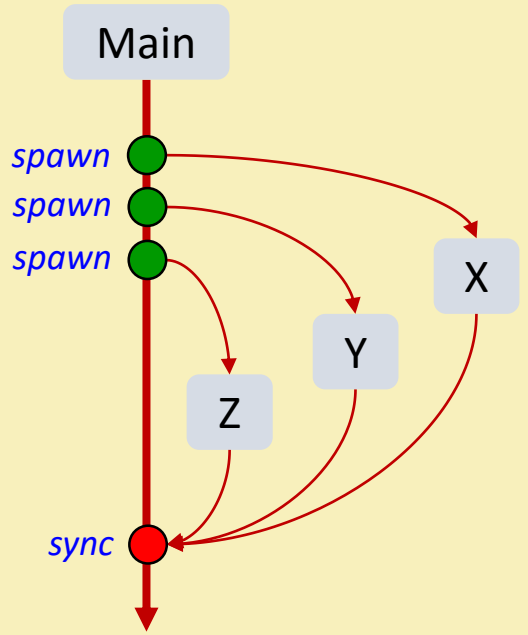
https://www.cilkplus.org

Intel Cilk Plus

- **Data parallelism**
  - ❑ Распараллеливание циклов: **_Cilk_for**
  - ❑ Конструкции для реализации редукции: Reducers
  - ❑ Векторная обработка массивов (Array notation) #pragma simd, elemental functions
- **Task parallelism**
  - ❑ Порождение задач (tasks): **_Cilk_spawn**
  - ❑ Синхронизация задач: **_Cilk_sync**
- Средствами Cilk Plus эффективно распараллеливаются:
  - алгоритмы типа “разделяй и властвуй” (divide & conquer): сортировка слиянием, быстрая сортировка и т.д.
  - операции над рекурсивными структурами данных (деревья, списки)

Intel Cilk Plus

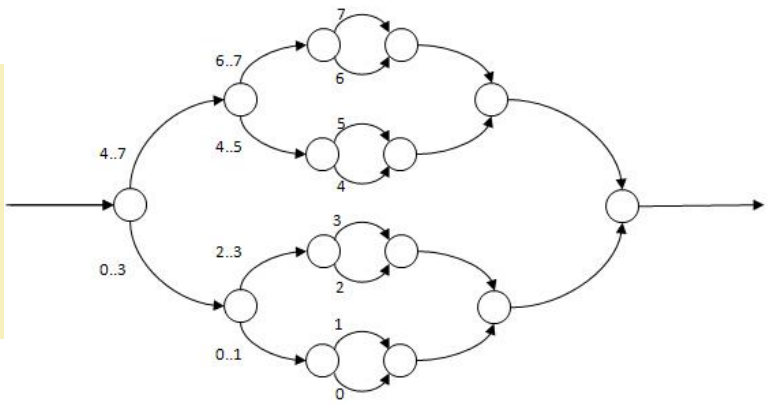
```
#include <iostream>
#include <cilk/cilk.h>

void handler(int id, char *name)
{
    std::cout << "Hello World form " << id << ": "
              << name << std::endl;
}

int main()
{

    cilk_spawn handler(100, "Task X");
    cilk_spawn handler(120, "Task Y");
    cilk_spawn handler(130, "Task Z");
    cilk_sync;

    return 0;
}
```



Intel Cilk Plus

```
int fib(int n)
{
    int x, y;
    if (n < 2)
        return n;
    x = cilk_spawn fib(n - 1);
    y = fib(n - 2);
    cilk_sync;
    return x + y;
}
```

```
#pragma cilk grainsize = 4
cilk_for(int i = begin; i < end; i += 2) {
    f(i);
}
```



Intel Cilk Plus

```
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>

unsigned int compute(unsigned int i)
{
    return i;
}

int main(int argc, char* argv[])
{
   unsigned long long int n = 1000000;
   // reducer с операцией +
   cilk::reducer_opadd<unsigned long long int> total(0);

   cilk_for (unsigned int i = 1; i <= n; ++i) {
     *total += compute(i);
   }

   std::cout << "Total " << total.get_value() << std::endl;
   return 0;
}
```

Intel Cilk Plus

- **Array Notation –** это языковые конструкции Intel Cilk Plus информирующие компилятор о возможности использовать векторные инструкции для обработки массивов (SIMD processing)

- **Обращение к части массива: base[first:length:stride]**
  - A[:] – весь массив
  - A[3:99] – элементы с 3 по 99
  - B[:][5] – столбец 5 матрицы
  - C[0:3][0:4] – подматрица

```
s[0:n] = sin(y[0:n])
```

```
if (a[0:n] < b[0:n])
    c[0:n] += 1;
else
    d[0:n] -= 1;
```

```
/* Reductions */
sum = __sec_reduce_add(a[0:n] * b[0:n]);
```

Intel Cilk Plus

```
void saxpy(float a, float *x, float *y, int n)
{
    #pragma simd
    for (int i = 0; i < n; i++) {
        y[i] += a * x[i];
    }
}
```

```
float sum(float *x, int n)
{
    float sum = 0;
    #pragma simd reduction(+:sum)
    for (int i = 0; i < n; i++) {
        sum += *x++;
    }
    return sum;
}
```

Intel Threading Building Blocks (Intel TBB)

Intel Threading Building Blocks

- **Intel Treading Building Blocks (TBB)** –
  это кроссплатформенная библиотека шаблонов C++ для создания многопоточных программ

- История развития:
  - 2006 – Intel TBB v1.0 (Intel compiler only)
  - 2007 – Intel TBB v2.0 (Open Source, GPLv2)
  - 2008 – Intel TBB v2.1 (thread affinity, cancellation)
  - 2009 – Intel TBB v2.2 (C++0x lambda functions)
  - …
  - 2011 – Intel TBB v4.0
  - 2012 – Intel TBB v4.1
  - **2016 – Intel TBB v4.4**

http://threadingbuildingblocks.org

Intel Threading Building Blocks

- Алгоритмы: `parallel_for, parallel_reduce, parallel_scan, parallel_while, parallel_do, parallel_pipeline, parallel_sort`
- Контейнеры: `concurrent_queue, concurrent_vector, concurrent_hash_map`
- Аллокаторы памяти: `scalable_malloc, scalable_free, scalable_realloc, scalable_calloc, scalable_allocator, cache_aligned_allocator`
- Мьютексы: `mutex, spin_mutex, queuing_mutex, spin_rw_mutex, queuing_rw_mutex, recursive mutex`
- Атомарные операции: `fetch_and_add, fetch_and_increment, fetch_and_decrement, compare_and_swap, fetch_and_store`
- Task-based parallelism (fork-join) + work stealing

Intel Threading Building Blocks

- Алгоритмы: `parallel_for, parallel_reduce, parallel_scan, parallel_while, parallel_do, parallel_pipeline, parallel_sort`
- Контейнеры: `concurrent_queue, concurrent_vector, concurrent_hash_map`
- Аллокаторы памяти: `scalable_malloc, scalable_free, scalable_realloc, scalable_calloc, scalable_allocator, cache_aligned_allocator`
- Мьютексы: `mutex, spin_mutex, queuing_mutex, spin_rw_mutex, queuing_rw_mutex, recursive mutex`
- Атомарные операции: `fetch_and_add, fetch_and_increment, fetch_and_decrement, compare_and_swap, fetch_and_store`
- Task-based parallelism (fork-join) + work stealing

Intel Threading Building Blocks

```
#include <iostream>
#include <tbb/task_scheduler_init.h>
#include <tbb/tick_count.h>
#include <tbb/parallel_for.h>
#include <tbb/blocked_range.h>

class saxpy_par {
public:
    saxpy_par(float a, float *x, float *y):
        a_(a), x_(x), y_(y) {}

    void operator()(const blocked_range<size_t> &r) const
    {
        for (size_t i = r.begin(); i != r.end(); ++i)
            y_[i] += a_ * x_[i];
    }

private:
    float const a_;
    float *const x_;
    float *const y_;
};
```

Intel Threading Building Blocks

```
int main()
{
    float a = 2.0;
    float *x, *y;
    size_t n = 100000000;

    x = new float[n];
    y = new float[n];
    for (size_t i = 0; i < n; ++i)
        x[i] = 5.0;

    tick_count t0 = tick_count::now();
    task_scheduler_init init(4);
    parallel_for(blocked_range<size_t>(0, n), saxpy_par(a, x, y),
                 auto_partitioner());
    tick_count t1 = tick_count::now();
    cout << "Time: " << (t1 - t0).seconds() << " sec." << endl;

    delete[] x;
    delete[] y;

    return 0;
}
```

Intel Threading Building Blocks

- Класс **blocked_range**(begin, end, grainsize) описывает одномерное пространство итераций
- В Intel TBB доступно описание многомерных пространств итераций (blocked_range2d, ...)

```
    x = new float[n];
    y = new float[n];
    for (size_t i = 0; i < n; ++i)
        x[i] = 5.0;

    tick_count t0 = tick_count::now();
    task_scheduler_init init(4);
    parallel_for(blocked_range<size_t>(0, n), saxpy_par(a, x, y),
                 auto_partitioner());
    tick_count t1 = tick_count::now();
    cout << "Time: " << (t1 - t0).seconds() << " sec." << endl;

    delete[] x;
    delete[] y;

    return 0;
}
```