# Intel Thread Building Blocks.

Sunday, June 9, 2013 3:21 PM

TBB knihovna jako taková využívá vláken OS, ale programátor s nimi již nepracuje. Místo toho jsou v TBB tzv. Tasky – knihovna pak vykonává jednotlivé úlohy paralelně. Lze si napsat vlastní plánovač tasků a protože TBB nespoléhá jen na direktivy překladače, lze paralelizovat i takové úlohy, které by se s OpenMP paralelizovaly těžko.

Viz příloha z přednášek.

http://www.slideshare.net/psteinb/abhishek-sync-locks-gdc

# Printout

Sunday, June 9, 2013 3:23 PM

```
  localMaxAbsDiff = max(localMaxAbsDiff,
                        sublocalMaxAbsDiff);
} //pragma omp parallel


if (cnt) {
  //if to avoid division by zero
  floattype tmp;
  tmp = valuescnt;
  tmp = 1.0/tmp;

  (*stats).AvgAbsDiff = localAvgAbsDiff*tmp;
  (*stats).AvgDiff = localAvgDiff*tmp;
  (*stats).MaxAbsDiff = localMaxAbsDiff;
}
```

# Intel Thread Building Blocks

- Open Source, podpora více platforem
  - Stejně jako OpenMP
- Ačkoliv si je TBB vědoma vláken poskytovaných OS, myšlenka je takový, že programátor už s vlákny nepracuje
- Namísto vláken specifikuje úlohy, tasks, a jejich návaznosti

- Knihovna vykonává úlohy paralelně, jak nejlépe to jde
  - Obsahuje různé optimalizace v jakém pořadí úlohy vykonávat
  - Ne vždy platí, že FIFO je nejlepší strategie
    - Využívá se technika „task stealing"
  - Lze si napsat vlastní plánovač

- Stejně jako STL, i TBB intenzivně používá C++ šablony
  - Tj. nelze ji použít v C jako OpenMP

# Page 2

Sunday, June 9, 2013 3:23 PM

- Základní konstrukce TBB jsou
  - parallel_for, parallel_reduce, parallel_scan
  - parallel_while, parallel_do,pipeline, parallel_sort
  - paralelní kontainery, které jsou v STL
  - mutex, spin_mutex, queuing_mutex, spin_rw_mutex, queuing_rw_mutex, recursive mutex
  - fetch_and_add, fetch_and_increment, fetch_and_decrement, compare_and_swap, fetch_and_store

- TBB dokáže použít vlastní paměťový manažer, který je optimalizovaný na paradigma výpočtu úloh

- TBB je náročnější se naučit, ale zase se to vyplatí na výkonu, pokud se začíná psát nová aplikace, než např. paralelizovat s OpenMP
  - Navíc TBB se nespoléhá na direktivy pro překladač, takže podmínka if OpenMP se dá realizovat libovolně složitá, nebo se dají udělat konstrukce, který by šli s OpenMP udělat jen velmi obtížně – např. cancellation výpočtu v OpenMP není

- Např. chceme-li spustit na pozadí několik výpočetně náročných úloh paralelně

```
tbb::task* CMasterCalculationTBBLogic::execute() {
 tbb::task_list list;

 for (int i=gaiFirst; i<=gaiLast; i++)
  list.push_back(*new(allocate_child()) CTask(i));

 set_ref_count(gaiLast-gaiFirst+2); //počet úloh +1
 spawn_and_wait_for_all(list);

 return NULL; //non-NULL by byla úloha, která by měla
}         //být spuštěna jako další/závislá na téhle
```

# Page 3

Sunday, June 9, 2013 3:23 PM

- Když bychom např. násobili vektor

```
class CMulVect {
    floattype *mA, *mB;
    int mLen;
public:
    floattype mProduct;

    CMulVect(floattype *a, floattype *b, int len) :
            mA(a), mB(b), mLen(len), mProduct(0.0) {}
    CMulVect(CMulVect& x, tbb::split) : mA(x.mA),
             mB(x.mB), mLen(x.mLen), mProduct(0.0) {}

    void join(const CMulVect& y) {
        mProduct += y.mProduct;
    }

    void operator()( const
                    tbb::blocked_range<size_t>& r ) {
        int r_end = r.end();

        //Taken from Intel's tutorial on TBB
        //by declaring these variables, we make it
         //obvious to the compiler that they
        //can be held in registers instead in memory
         // => speedup

        floattype *a = mA;
        floattype *b = mB;
        floattype sum = 0.0;

        for (int i=r.begin(); i!=r_end; ++i)
            sum += a[i]*b[i];

        mProduct += sum;
    }
};
```

# Page 4

Sunday, June 9, 2013 3:23 PM

```
floattype MulVect(floattype *a, floattype *b,
                  int len) {

    floattype result = 0.0;

    //Jsou data tak velká, aby se je vůbec
    //vyplatilo počítat paralelně?
    if (len<=rmSerialThreshold) {
       for (int i=0; i<len; i++)
          result += a[i]*b[i];
    } else {
        CMulVect muller(a, b, len);

        tbb::parallel_reduce(
          tbb::blocked_range<size_t>(0,len), muller);
        result = muller.mProduct;
    }
    return result;
}
```

# Případová studie použití TBB

- Uvažujme výpočetně náročnou aplikaci nad rozsáhlými daty s GUI
- Jedno vlákno bude obsluhovat GUI
- Jedno vlákno bude obsluhovat I/O
  - Naivní přístup je jedno vlákno pro zápis a jedno pro čtení
  - Lepší je jenom jedno vlákno a použití asynchronních I/O operací
    - OS si je uspořádá sám pro lepší výkon
    - Např. disky mají Native Command Queuing
  - Ale co s výpočetní částí?
    - Určitě v tom bude alespoň jedno vlákno, aby výpočet běžel na pozadí a GUI bylo responsive

# Page 5

Sunday, June 9, 2013 3:23 PM

- Přístup s psaním vláken by znamenal postarat se
  - Vytváření a rušení vláken, což je další režie pro OS
  - Jejich správnou synchronizaci, což je náročné, jakmile se program stává složitější
  - Výkonnostně je rozdíl, jestli se synchronizuje v kernel-mode, nebo v user-mode address space
  - Výkonnostně hraje roli, zda se k datům přistupuje ze stejného procesoru – thread affinity

  - Optimální počet vláken odpovídá počtu procesorových jader v systému, což ale není přístup, který je vždy možný při psaní vláken
    - Např. počet vláken může odpovídat tomu, jaká je metoda výpočtu – problém škálovatelnosti
      - S TBB se taková vlákna nahradí úlohami

- S TBB
  - O výše uvedené problémy se TBB postará
  - Programátor napíše jedno vlákno, ve kterém pak spustí výpočet úloh TBB
  - Nicméně programátor si stále musí být vědom toho, jak se píší paralelní programy s vlákny, aby mohl správně používat synchronizační primitiva TBB
    - Optimálně se naprogramují pouze úlohy s tím, že každá úloha po dokončení vrátí další úlohu, která se má vykonat jako navazující
    - Jako bariéra na dokončení více úloh se pak použije task list
    - Úlohy však mohou sdílet proměnné, a proto je třeba znát teorii o psaní vláken
      - Aby byla představa, že 2 úlohy mohou, ale i nemusí, běžet paralelně, např. když se má správně použít mutex, podmínka...