Παράλληλος Προγραμματισμός με τα Threading Building Blocks

Συστήματα Παράλληλης Επεξεργασίας
9ο Εξάμηνο, ΣΗΜΜΥ

Εργ. Υπολογιστικών Συστημάτων
Σχολή ΗΜΜΥ, Ε.Μ.Π.

National Technical University of Athens
CSLab

Νοέμβριος 2019

Threading Building Blocks

Εισαγωγή

- **C++ template library** για αποδοτικό και εύκολο παράλληλο προγραμματισμό σε πλατφόρμες μοιραζόμενης μνήμης

- Αναπτύσσεται από την Intel από το 2004 (open-source από το 2007)
- Δεν είναι καινούργια γλώσσα ή επέκταση
- Μεταφέρσιμη στους περισσότερους C++ compilers, λειτουργικά συστήματα και αρχιτεκτονικές
- Παρέχει ένα πλούσιο σύστημα από παραμετροποιήσιμα (templated)"ready to use" αλγοριθμικά μοτίβα και δομές (αντίστοιχα με τη C++ STL)

- Εκμεταλλεύεται τα **lambda expressions** της C++11 από το 2008 (gcc>=4.5)

Βασικά Χαρακτηριστικά

Tasks

- Ο προγραμματιστής ορίζει **tasks** αντί για threads
  - Επικεντρώνεται στην **έκφραση** του παραλληλισμού
  - Η βιβλιοθήκη **υλοποιεί** τον παραλληλισμό:
    - διάσπαση συνολικής δουλειάς σε επιμέρους εργασίες
    - δρομολόγηση εργασιών
    - συγχρονισμός
    - ισοκατανομή φορτίου
    - διαχείριση πόρων συστήματος και εσωτερικών μηχανισμών

Βασικά χαρακτηριστικά

Κλιμακωσιμότητα

- Είναι σχεδιασμένη για **κλιμακωσιμότητα**
  - Η συνολική δουλειά σπάει σε πολλά μικρά κομμάτια (tasks), συνήθως πολύ περισσότερα από τον αριθμό των επεξεργαστών (“**parallel slack**”)
  - Εξασφαλίζεται ότι πάντα θα υπάρχει διαθέσιμη δουλειά για κάθε **επιπλέον επεξεργαστή** που προστίθεται
  - Ο μηχανισμός **load balancing** εξασφαλίζει την κλιμακώσιμη απόδοση

Βασικά χαρακτηριστικά

Γενικευμένος προγραμματισμός

- Εκμεταλλεύεται τη δύναμη και την ευελιξία του **γενικευμένου προγραμματισμού**
  - Παρέχει ένα πλούσιο σύνολο από *παραμετροποιήσιμα* (templated), “ready to use” παράλληλα αλγοριθμικά μοτίβα και δομές
    - Αντίστοιχα με την C++ STL για τα σειριακά προγράμματα
  - δεν απαιτεί ειδική υποστήριξη από το μεταγλωττιστή

Threading Building Blocks

TBB 4.0 Components

Generic Parallel Algorithms

parallel_for
parallel_reduce
parallel_scan
parallel_do
pipeline, parallel_pipeline
parallel_sort
parallel_invoke

Concurrent Containers

concurrent_unordered_map
concurrent_unordered_set
concurrent_unordered_hash_map
concurrent_queue
concurrent_bounded_queue
concurrent_priority_queue
concurrent_vector

Synchronization Primitives

atomic
mutex, recursive_mutex
spin_mutex, spin_rw_mutex
queueing_mutex, queueing_rw_mutex

Raw Tasking

task
task_group
task_list
task_scheduler_observer

Flow Graph

graph
function_node
broadcast_node
...

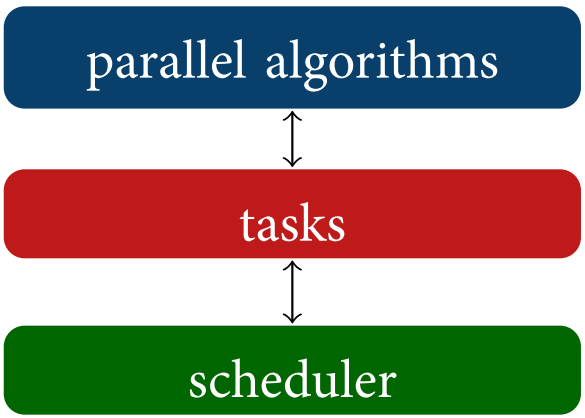
Memory Allocation

tbb_allocator, cache_aligned_allocator, scalable_allocator

Threading Building Blocks

Tasks

- Εκφράζουν μια στοιχειώδη ανεξάρτητη εργασία στο πρόγραμμα του χρήστη
  - πολύ πιο lightweight από τα native threads του λειτουργικού (user-level, small-sized, non-preemptible, short-lived)



- Δυνατότητα άμεσης χρήσης των tasks από τον προγραμματιστή
  - δημιουργία αυθαίρετα πολύπλοκων γράφων διεργασιών

Tasks

Προγραμματιστικό Μοντέλο

- Δύο βασικές λειτουργίες για την περιγραφή ενός task graph
  - *spawn*: δημιουργία εργασίας
  - *wait*: συγχρονισμός εργασιών

Threading Building Blocks

Task Objects (Templates)

```
long SerialFib(long n) {
  if (n<2)
    return n;
  else
    return SerialFib(n-1)
         +SerialFib(n-2);
}
```

```
long n, sum;

FibTask& r = *new (
  allocate_root())
  FibTask(n,&sum);
spawn_root_and_wait(r);

cout << sum;
```

```
class FibTask: public task {
  const long n;
  long *const_sum;
  FibTask(long n_, long* sum_) {
    n=n_; sum=sum_;
  }
  task* execute() {
    if (n < cutOff) *sum = SerialFib(n);
    else {
      long x,y;
          FibTask& a = *new (allocate_child()) FibTask(n-1,&x);
          FibTask& b = *new (allocate_child()) FibTask(n-2,&y);
      set_ref_count(3);
      spawn(b);
      spawn(a);
      wait_for_all();
      * sum = x+y;
    }
    return NULL;
  }
}
```

Threading Building Blocks

Tasks (Lambdas)

```
long SerialFib(long n) {
  if (n<16)
    return n;
  else {
      int x,y;
      tbb::task_group g;
      g.run ( [&] { x  = ParallelFib(n-1); } );
      g.run ( [&] { y  = ParallelFib(n-2); } );
      g.wait();
      return x+y;
  }
}
```

Σκονάκι στα lambdas της C++

```
[capture_mode](formal_parameters)->return_type{body}
```

- [capture_mode]: Τρόπος περάσματος παραμέτρων
  - [=] : by value
  - [&] : by reference
  - [] : no capture
- (formal_parameters): Μπορεί να παραληφθεί αν δεν υπάρχουν παράμετροι και ο τύπος επιστροφής είναι implicit
- return_type: Μπορεί να παραληφθεί αν είναι void ή το body είναι τύπου "return expr"

Παραδείγματα:

```
[&](float x){sum+=x;}
```

```
[]{return rand();}
```

```
[&]{return *p++;}
```

```
[](float x, float y)->float{
    if (x<y) return x;
    else return y;
```

Threading Building Blocks

Αρχικοποίηση

- Για τη χρήση οποιουδήποτε παράλληλου αλγορίθμου της βιβλιοθήκης απαιτείται η δημιουργία ενός αντικειμένου *task_scheduler_init*

```
#include <tbb/task_scheduler_init.h>
#define NPROCS 4

int main() {
    tbb::task_scheduler_init init(NPROCS);
    ...
}
```

Παραλληλοποίηση for-loop

parallel_for

- Δήλωση

```
#include <tbb/tbb_blocked_range.h>
#include <tbb/parallel_for.h>

template <typename Range, typename Body>
Body parallel_for (const Range& r, const Body& b);
```

- Η `parallel_for` αναλαμβάνει:
  - Να διασπάσει το αρχικό range σε πολλά μικρότερα
    - Η βιβλιοθήκη παρέχει τις κλάσεις *blocked_range*, *blocked_range2d*, *blocked_range3d*
  - Να εφαρμόσει την ανώνυμη συνάρτηση σε κάθε subrange

Παραλληλοποίηση for-loop

Παράδειγμα

- Σειριακός κώδικας:

```
float a[1000];
for ( int i = 0  i !=1000; ++i )
    foo(a[i]);
```

- Παράλληλος κώδικας (lambdas):

```
tbb::parallel_for(
    tbb::blocked_range<size_t>(0,1000),
    [=](const tbb::blocked_range<size_t>& r) {
        for (size_t i = r.begin() ; i != r.end(); ++i )
            foo(a[i]);
    }
);
```

- tbb::blocked_range<size_t>(0,1000) : δημιουργεί ανώνυμο αντικείμενο για την περιγραφή του αρχικού χώρου επαναλήψεων
- Η ανώνυμη συνάρτηση περιγράφει τι δουλειά θα γίνει σε οποιονδήποτε υποχώρο επαναλήψεων του loop

Threading Building Blocks

parallel_for και partitioners

```
parallel_for (blocked_range<size_t>(0,n,G), []() ...,
some partitioner())
```

- Δύο optional ορίσματα σε loop partitioners:
  - **Chunking**: το μέγεθος στον οποίο σταματά η αναδρομική διάσπαση
  - **Τύπος partitioner**

Chunking:

- Grainsize G: το ”κομμάτι” δουλειάς που θα αναλάβει κάθε task
  - Optional argument στον constructor του blocked_range

Partitioners:

- simple_partitioner: recursive binary splitting
- affinity_partitioner: ανάθεση με τρόπο ώστε να μεγιστοποιείται το cache locality
- auto_partitioner: αυτόματη επιλογή grainsize με ευριστική μέθοδο και ελαχιστοποίηση του range splitting, σε σημείο που να εξασφαλίζεται καλό load balancing

Load balancing

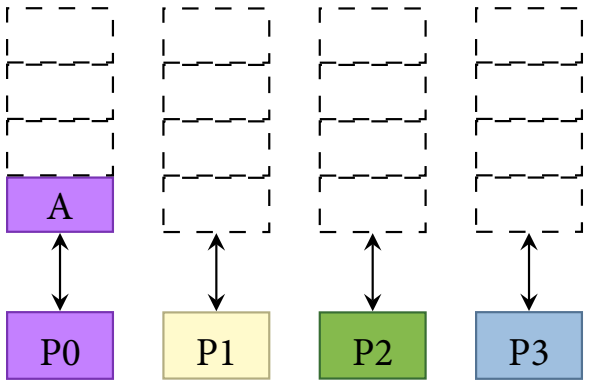
Το runtime των TBBs υλοποιεί δύο μηχανισμούς για load-balancing:

1. **Work stealing**: Εξασφαλίζει ισοκατανομή φορτίου
2. **Recursive splitting**: Επιτρέπει την επεξεργασία πάνω σε αυθαίρετα μικρά κομμάτια και τη βέλτιστη εκμετάλλευση της cache

parallel_for

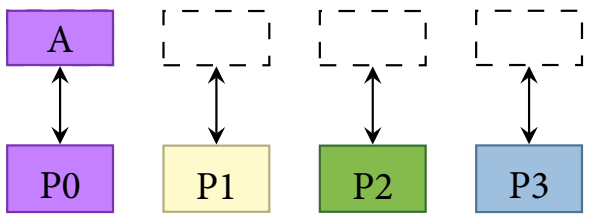
Work Stealing και Recursive Splitting

- Με parallel_for στον πίνακα A το *range* του πίνακα διασπάται αναδρομικά μέχρι να γίνει $\geq$*Grainsize*
- Σε κάθε εκτέλεση της αναδρομής το range διασπάται σε δύο subranges και δημιουργούνται 2 tasks
- Τα *worker threads* διατηρούν *double-ended queues* με τα tasks
- Κάθε worker παίρνει το task από τη *βάση της ουράς* και το **εκτελεί**
- Αν δεν το βρει, τότε **κλέβει** κάποιο από την *κορυφή της ουράς* ενός τυχαίου worker



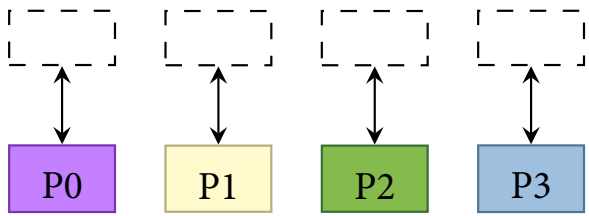
parallel_for

Work Stealing και Recursive Splitting



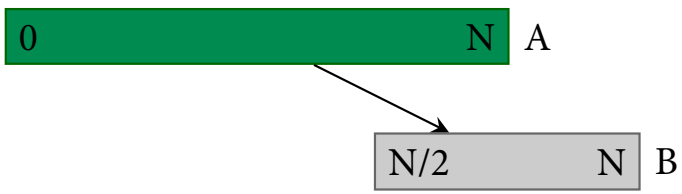
parallel_for

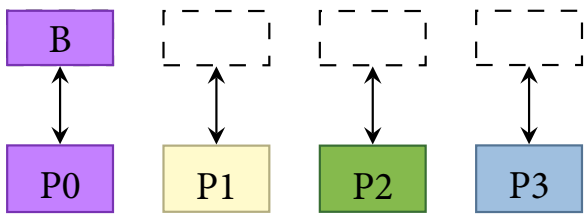
Work Stealing και Recursive Splitting



parallel_for

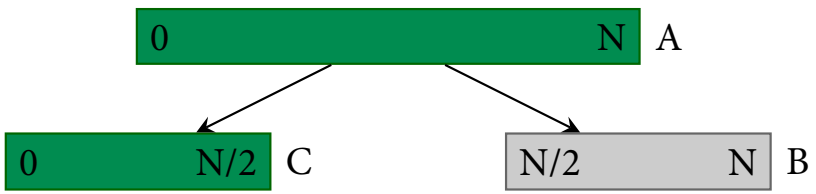
Work Stealing και Recursive Splitting

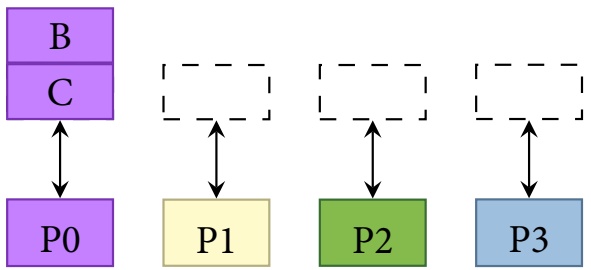




parallel_for

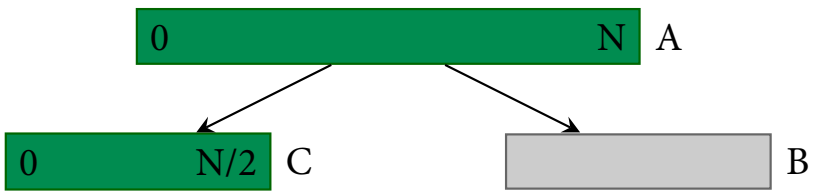
Work Stealing και Recursive Splitting

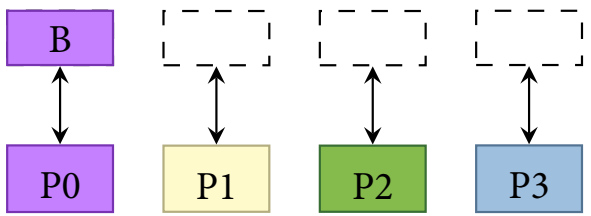




parallel_for

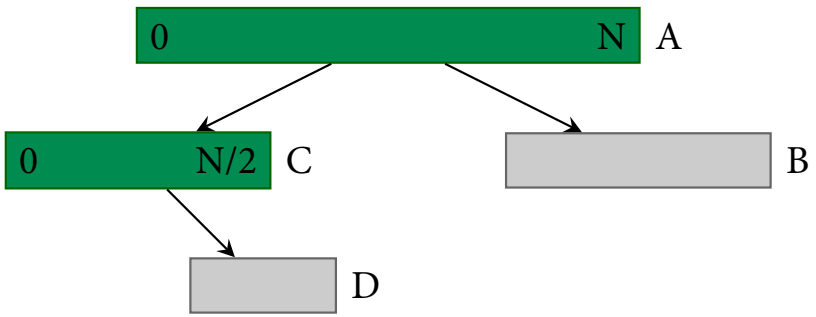
Work Stealing και Recursive Splitting

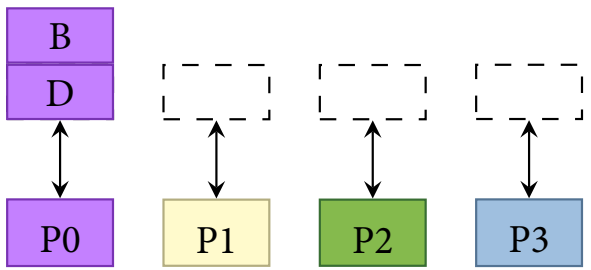




parallel_for

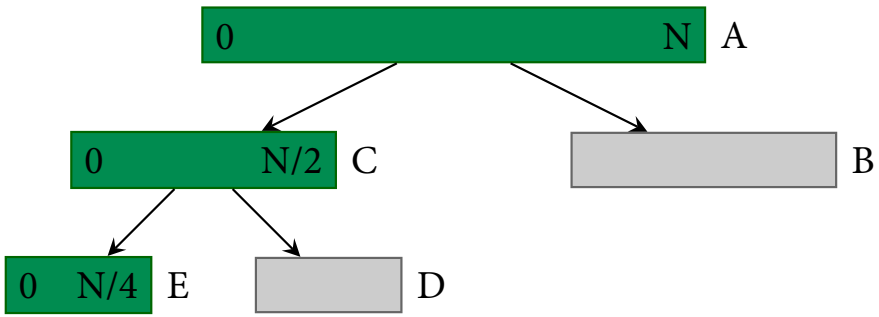
Work Stealing και Recursive Splitting

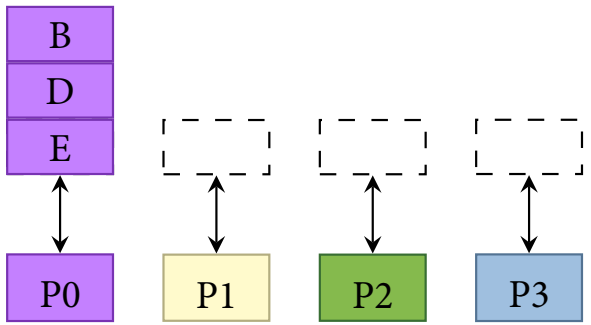




parallel_for

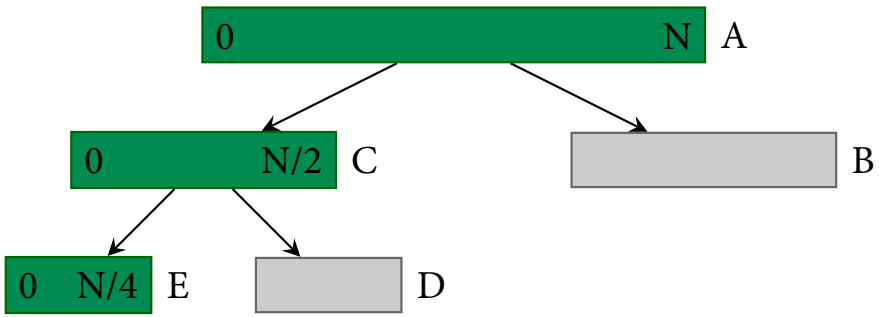
Work Stealing και Recursive Splitting

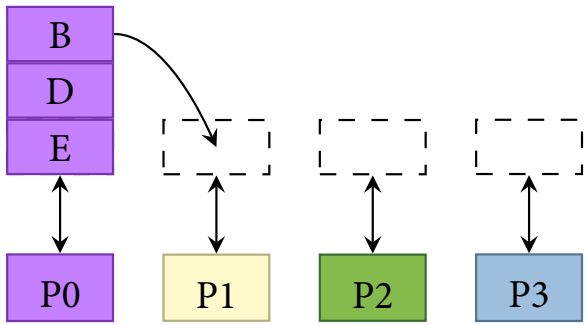




parallel_for

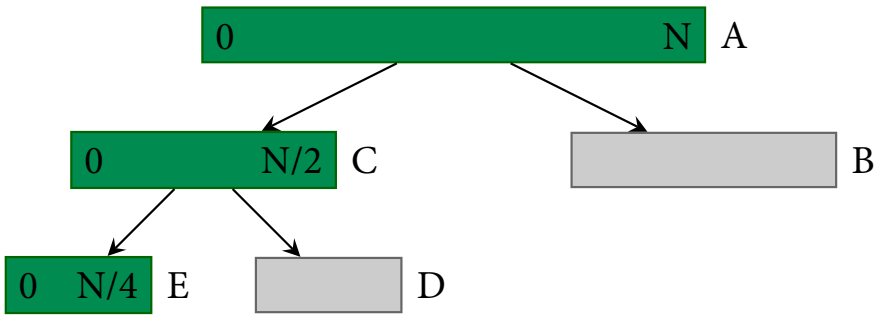
Work Stealing και Recursive Splitting

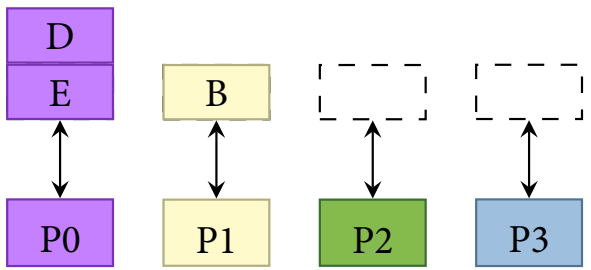




parallel_for

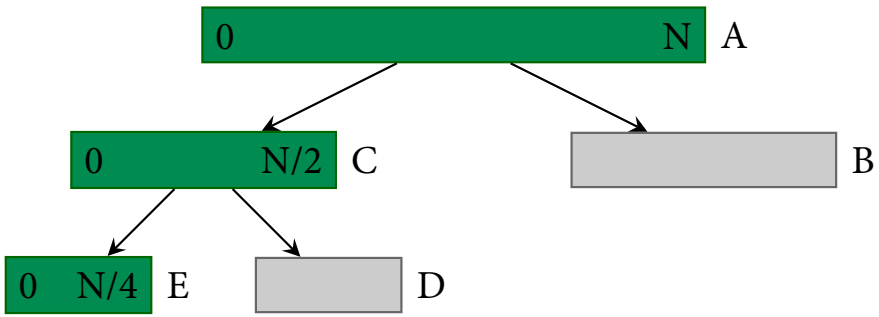
Work Stealing και Recursive Splitting

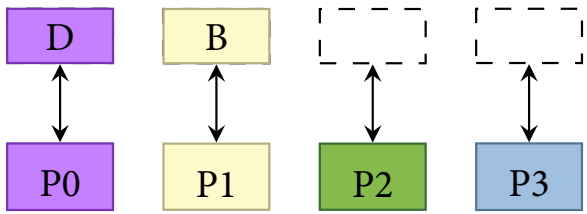




parallel_for

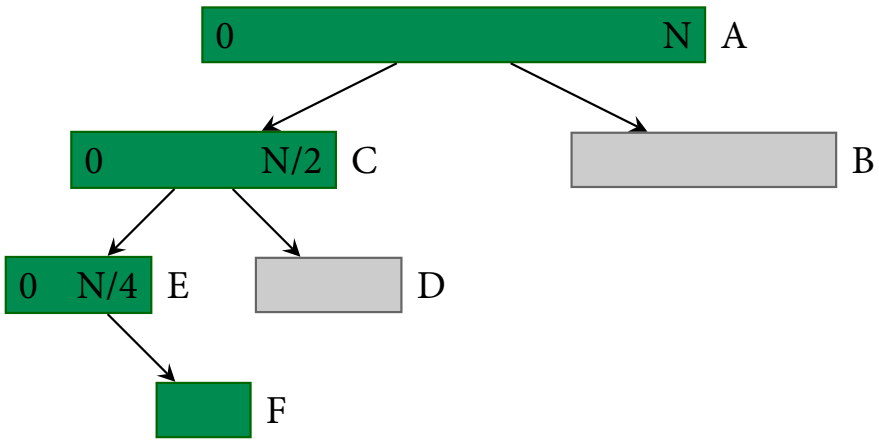
Work Stealing και Recursive Splitting

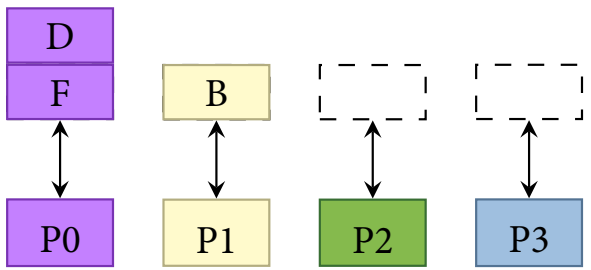




parallel_for

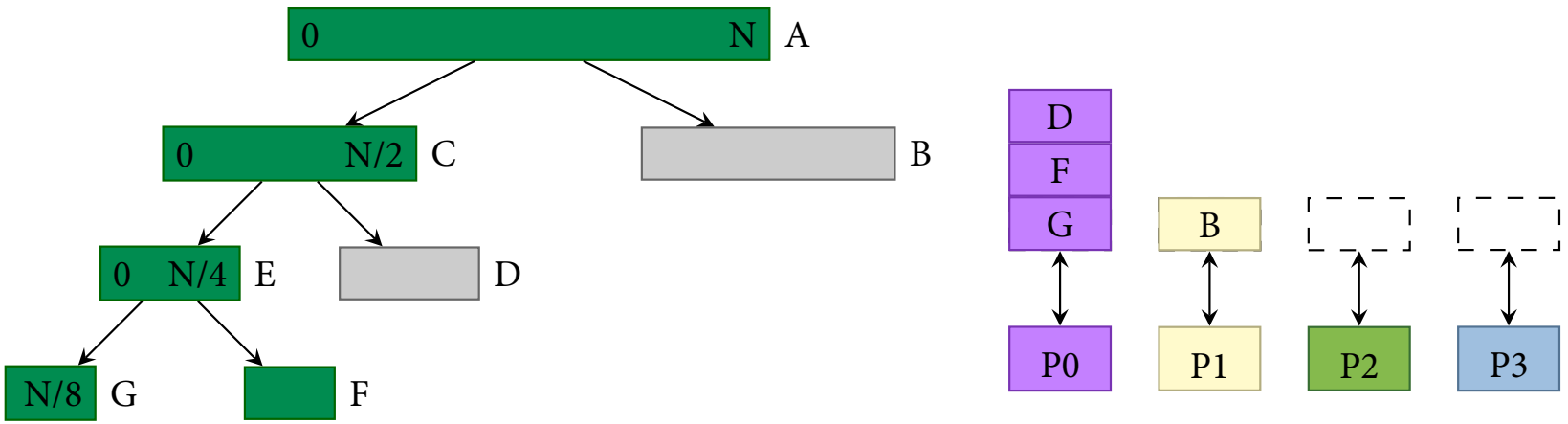
Work Stealing και Recursive Splitting





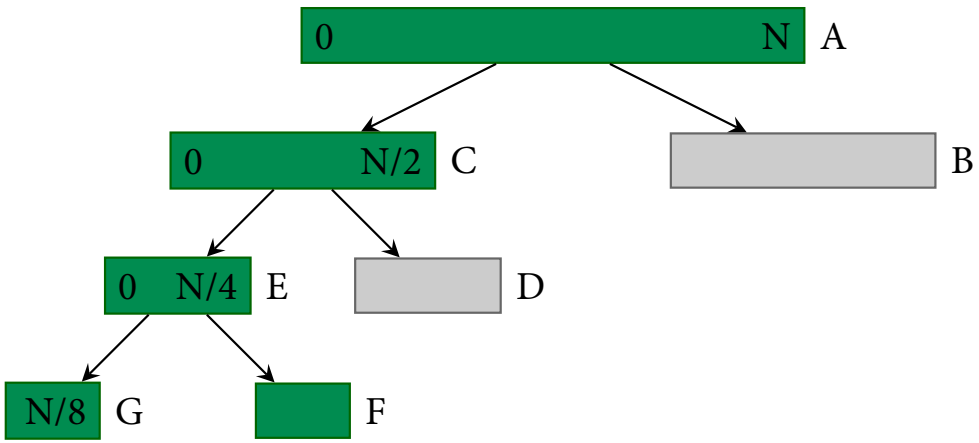
parallel_for

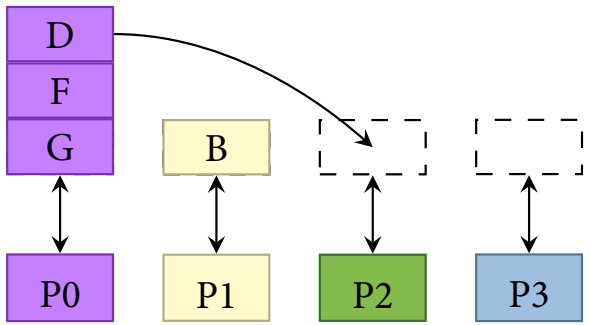
Work Stealing και Recursive Splitting



parallel_for

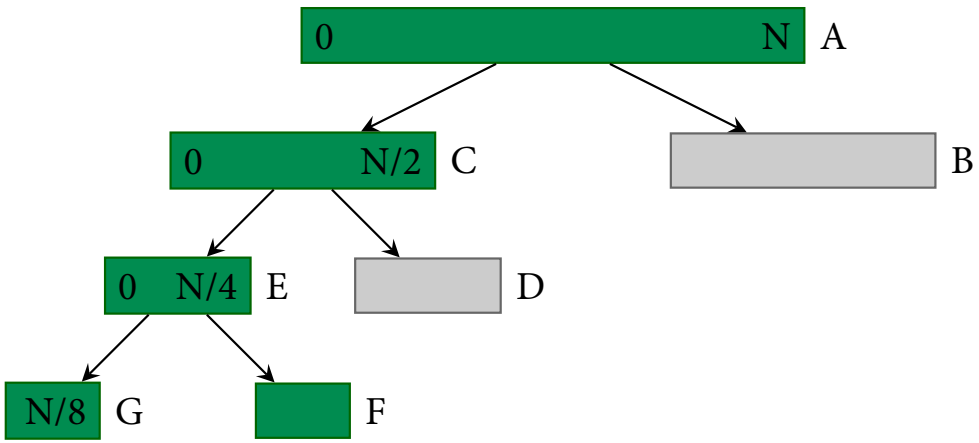
Work Stealing και Recursive Splitting

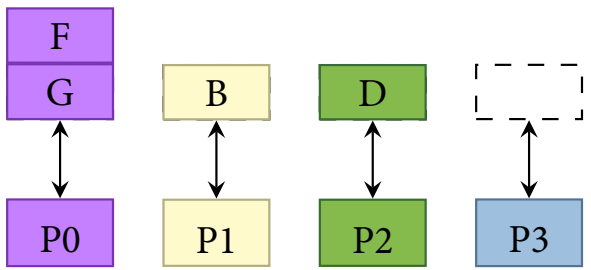




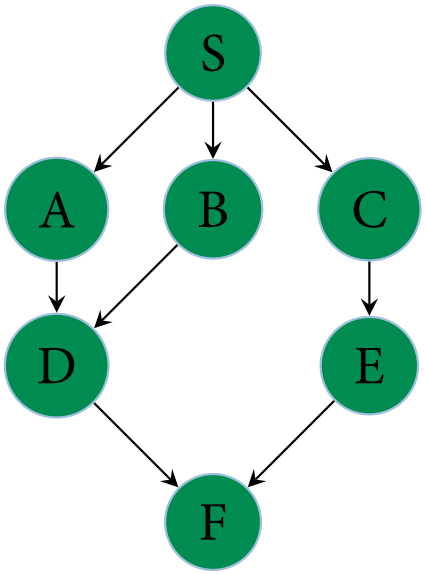
parallel_for

Work Stealing και Recursive Splitting

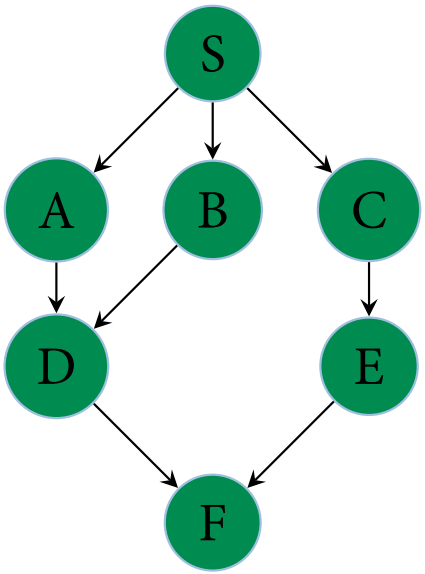




Task groups

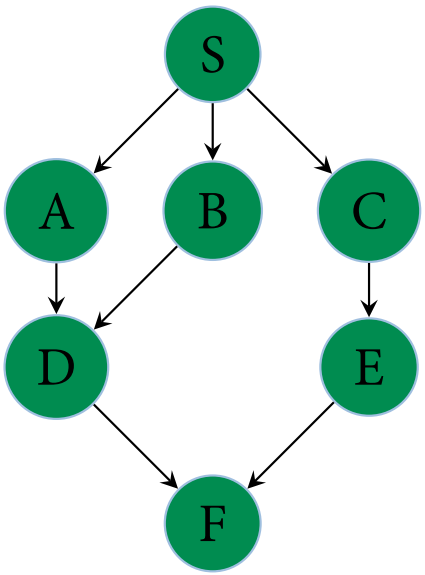


Nested Task Groups




```
S();
task_group g;
g.run ( [&] {  C(); E(); });
g.run ( [&] {
  task_group g1;
  g1.run ( [&] { A(); } );
  g1.run ( [&] { B(); } );
  g1.wait();
  D();
});
g.wait();
F();
```

Generic Task Graphs - Flow graph



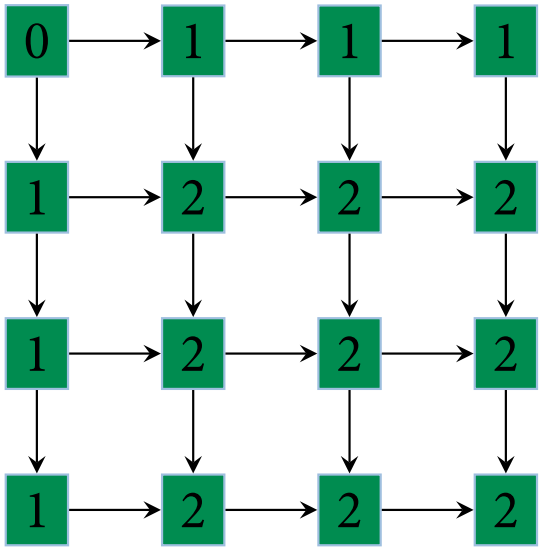

```
graph g;
broadcast_node <continue_msg> s;

continue_node <continue_msg> a(g,A());
continue_node <continue_msg> b(g,B());
continue_node <continue_msg> c(g,C());
continue_node <continue_msg> d(g,D());
continue_node <continue_msg> e(g,E());
continue_node <continue_msg> f(g,F());

make_edge(s,a);
make_edge(s,b);
make_edge(s,c);
make_edge(a,d);
make_edge(b,d);
make_edge(c,e);
make_edge(d,f);
make_edge(e,f);

S();
s.try_put(continue_msg());    //fire!
g.wait_for_all();
```

Task Objects + Reference counts




```
MeshTask* Mesh[4][4];

//for all tasks in mesh:
//allocate
//initialize south, east pointers
//set reference counters

Mesh[3][3]->
spawn_and_wait_for_all(*Mesh[0][0]
//execute last task
Mesh[3][3]->execute();
```

```
class MeshTask: public task {
  public:
    const int i,j; //coordinates
    MeshTask * south, east;

  task* execute() {
 double north_val = (i==0)? 0 : A[i-1][j];
  double west_val = (j==0)? 0 : A[i][j-1];
   A[i][j]=do_work(north_val,west_val);

    if (south!=NULL)
     if (!south->decrement_ref_count())
        spawn(*south);

    if (east!=NULL)
      if (!east->decrement_ref_count())
        spawn(*east);
    return NULL;
  }
}
```

Χρήσιμα links για τα TBBs

- Home http://www.threadingbuildingblocks.org/
- Documentation http://www.threadingbuildingblocks.org/documentation
- Forum http://software.intel.com/en-us/forums/intel-threading- building-blocks/