wsswift

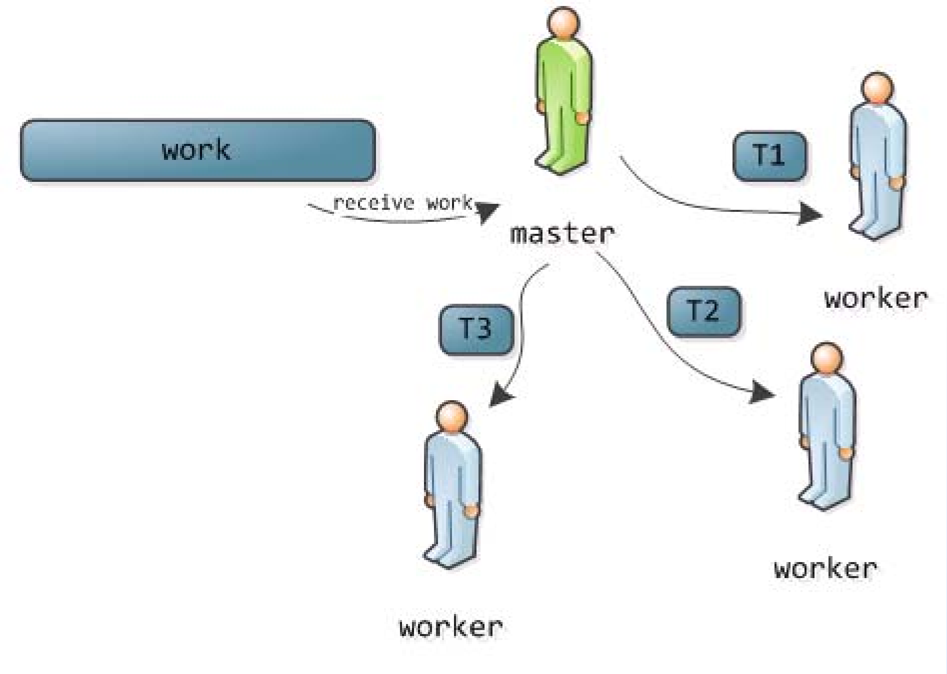
High-speed scheduling

Ionuț ROŞOIU, SCPD
Teodor CRIVĂȚ, SCPD

Agenda

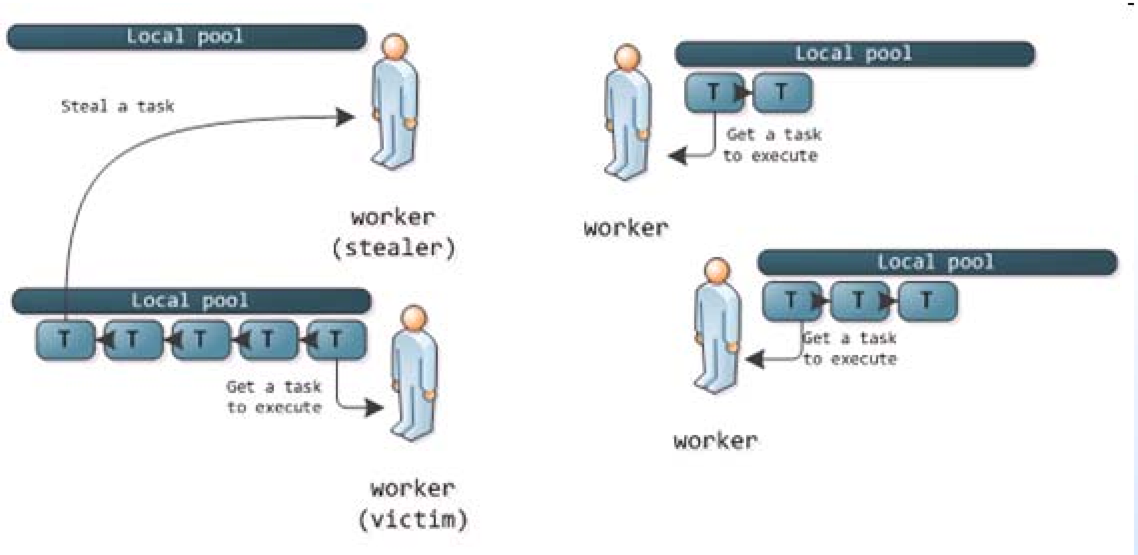
1. Task scheduling
2. Workstealing policy
3. Scheduler internals
4. Test results
5. Conclusions

1. Task scheduling



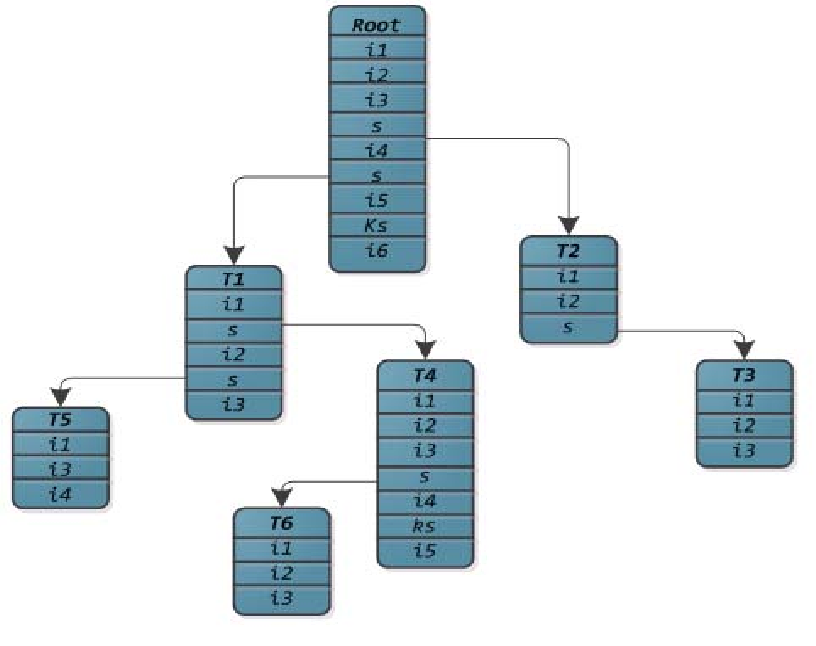
- **the traditional approach is master-worker**
- **synchronization overhead**
- **the master doesn't do useful work**

2. Workstealing policy



- no difference between master & slave
- communication overhead is low
- #steals << #tasks

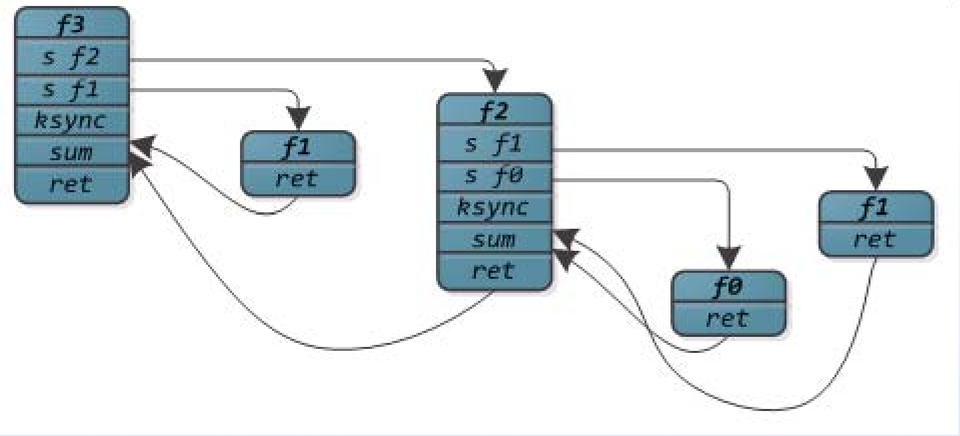
2. Workstealing policy



- **task creation → spawn tree**
- **flow dependencies**
- **data dependencies**

2. Workstealing policy

```
int Fibo(int n) {
        if (n < 2) {
                return n;
        } else {
                return Fibo(n-1) + Fibo(n-2);
        }
}
```



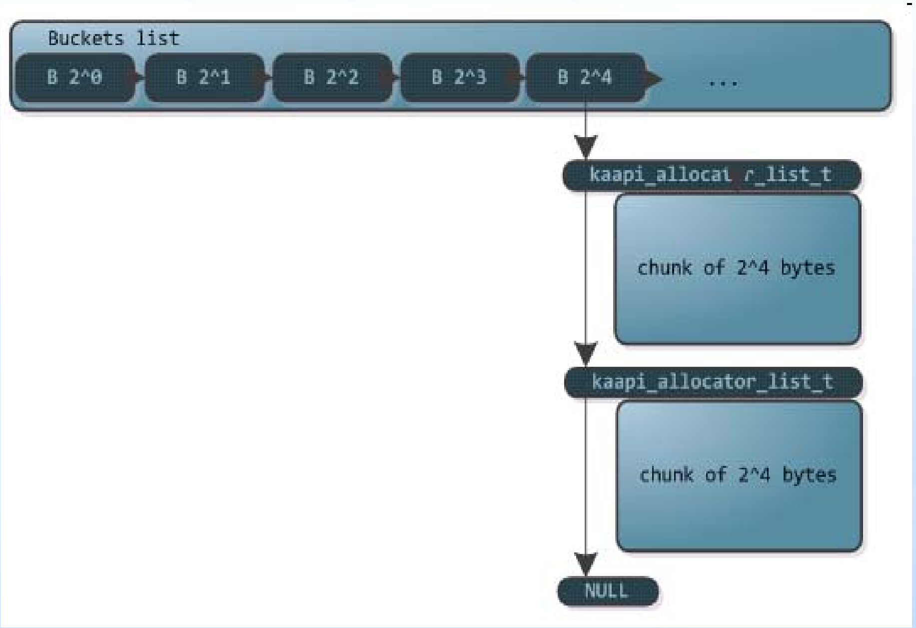
- **data dependency:** ***the sum***
- **flow dependency:** ***the return statement***

3. Scheduler internals

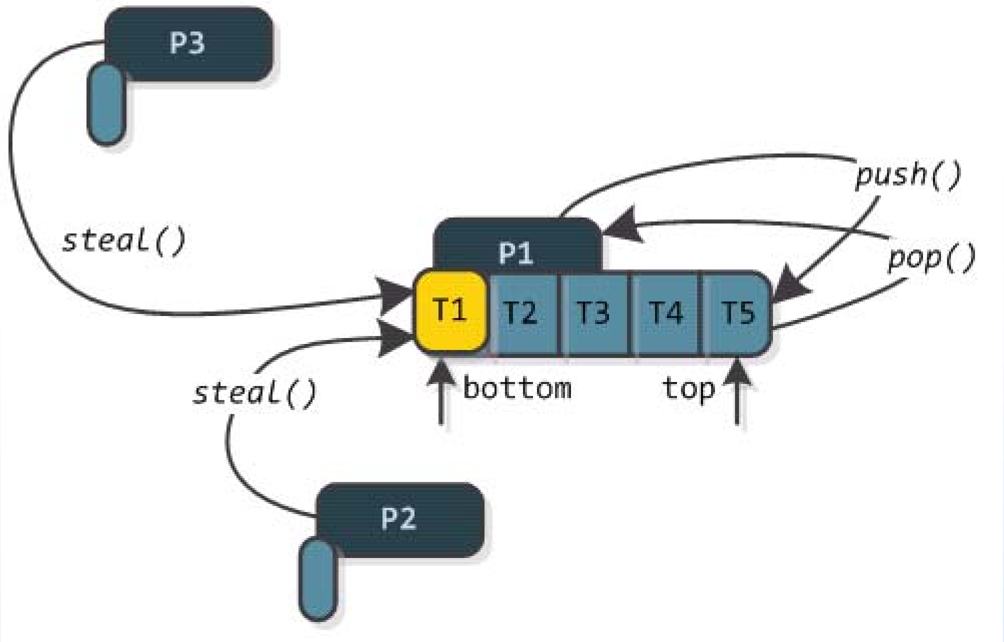
- **no locking! (only atomic increments and CAS)**

```
int compare_and_swap (int *word, int testval, int newval)
{
   int oldval;
   oldval = *word;
   if (oldval == testval) *word = newval;
   return oldval;
}
```

- **memory allocation**
  - malloc
  - Hoard parallel allocator
  - own allocator (buddy)

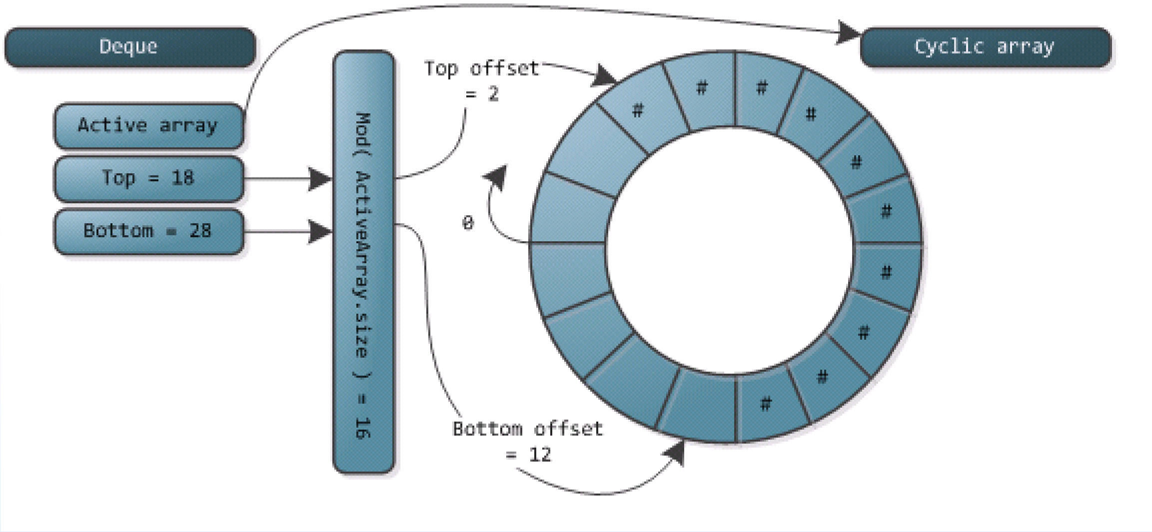


3. Scheduler internals



- **Owner operations: *push(), pop()***
- **Stealer operations: *steal()***
- **Data locality**

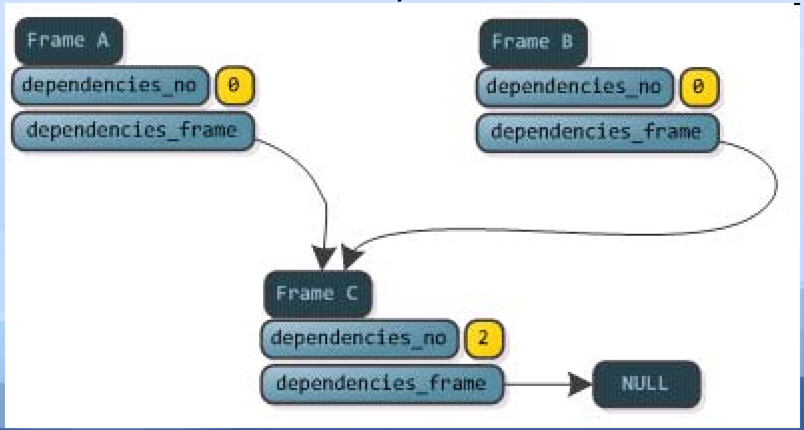
3. Scheduler internals



- the workqueue implementation is vital!
- based on Chase & Lev idea for Java
- the JavaVM memory model
- memory fences
- garbage collection

3. Scheduler internals

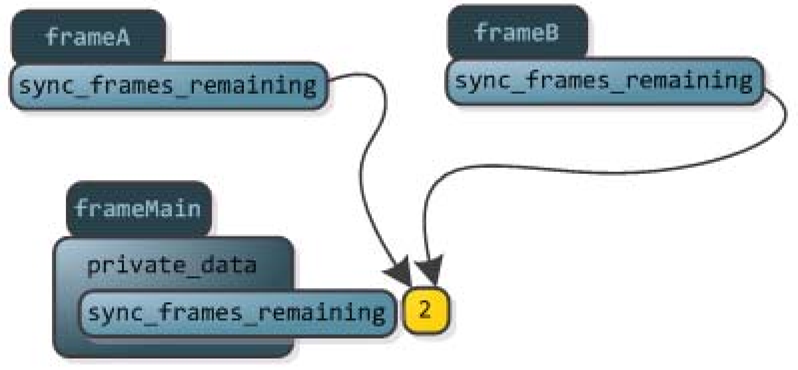
```
struct swift_frame {
      volatile int flags;     /*< frame flags */

#ifdef LOGGING_ON
      volatile int info;      /*< TODO: frame info (temporary, for debug) */
      int dbg;
      swift_id_t creator_id;  /*< the id of the initial creator for the frame */
#endif

      swift_closure_handler closure;      /*< the closure for this frame */

      volatile int dependencies_no;             /*< the number of unavailable variables */
      struct swift_frame *dependencies_frame;   /*< the frame that awaits the unavailable data */

      // closure-dependent data
      void *private_data;     /*< private data related to each specific closure
                              this gets deallocated when the frame is retired
                               */

      swift_size_t *sync_frames_remaining;
            /*< the number to decrement when finishing execution of this frame */

      // doubly-linked list
      struct swift_frame *prev;
      struct swift_frame *next;
};
```



3. Scheduler internals

- **sort-of "busy-wait" for flow dependencies**

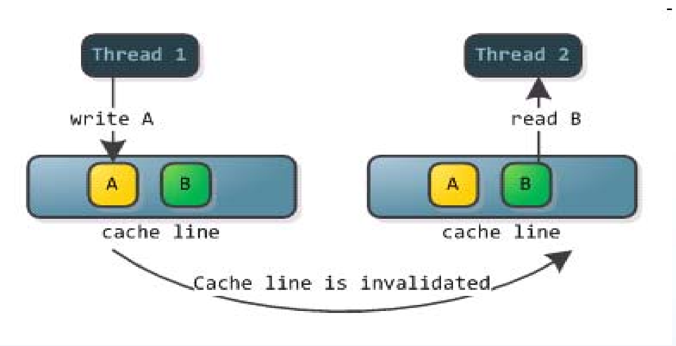
```
while ((n = SWIFT_ATOMIC_READ(data->sync_frames_remaining)))  {
            swift_scheduler_execute(thread, &status);
}
```

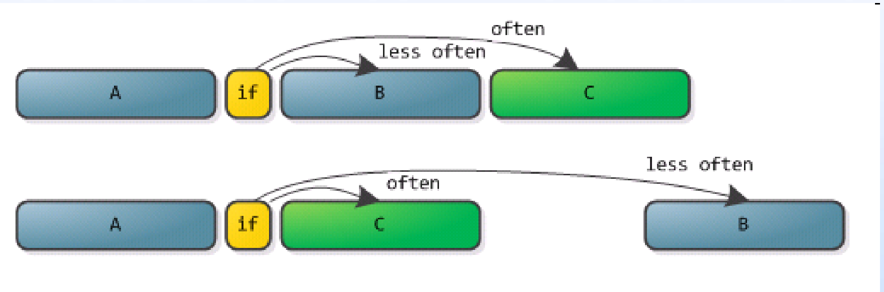


- **passing parameters**

```
typedef struct qs_data {
      int *a;
      int l;
      int r;

      char _pad1[SWIFT_CACHE_LINE_SIZE - sizeof(swift_size

      swift_size_t sync_frames_remaining;

      char _pad2[SWIFT_CACHE_LINE_SIZE - sizeof(swift_size_t)];
} qs_data_t;

-----------------------------------------------------------------

typedef struct fibo_data {
  int n;
  int *r;

  char _pad1[SWIFT_CACHE_LINE_SIZE - sizeof(swift_size_t)];

  swift_size_t sync_frames_remaining;

  char _pad2[SWIFT_CACHE_LINE_SIZE - sizeof(swift_size_t)];
} fibo_data_t;
```

3. Scheduler internals - optimizations





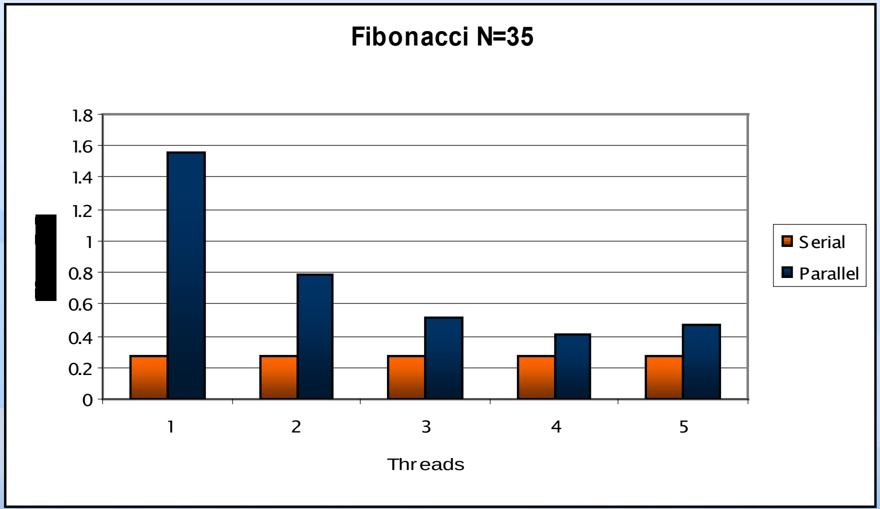
- **false sharing**
- **block reordering**
- **GCC flags**

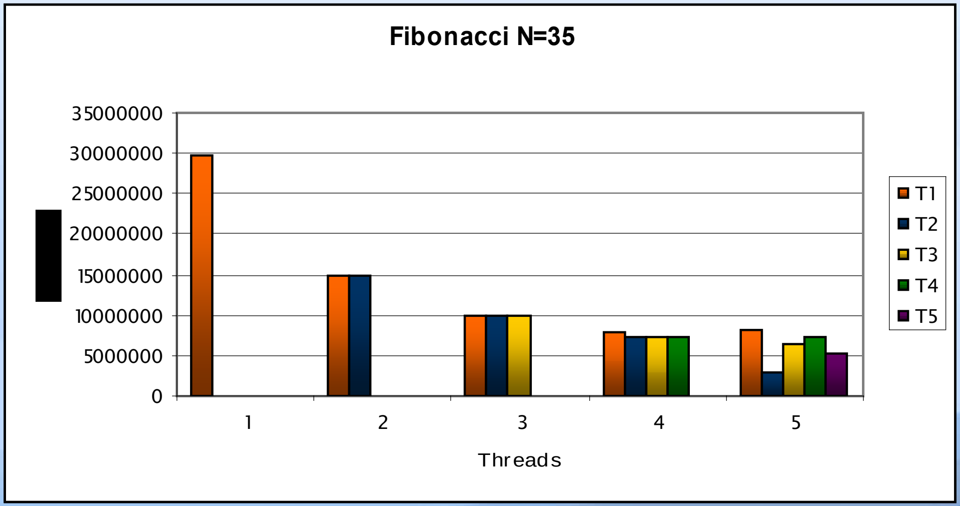
```
#define likely(x)       __builtin_expect((x),1)
#define unlikely(x)     __builtin_expect((x),0)
__attribute__((hot))
```

4. Test results – task scheduling cost

Intel Pentium Quad Core

```
$ cat /proc/cpuinfo |egrep 'name|MHz|cache size|bogo'
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
cpu MHz         : 2400.136
cache size      : 4096 KB
bogomips        : 4800.27
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
cpu MHz         : 2400.136
cache size      : 4096 KB
bogomips        : 4800.50
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
cpu MHz         : 2400.136
cache size      : 4096 KB
bogomips        : 4800.44
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
cpu MHz         : 2400.136
cache size      : 4096 KB
bogomips        : 4800.46
```





4. Test results

Intel Pentium Xeon

```
$ cat /proc/cpuinfo |egrep 'name|MHz|cache size|bogo'
model name      : Intel(R) Xeon(R) CPU           E5405  @ 2.00GHz
cpu MHz         : 2000.117
cache size      : 6144 KB
Bogomips        : 4000.23
   ... 7 more like this ...
```

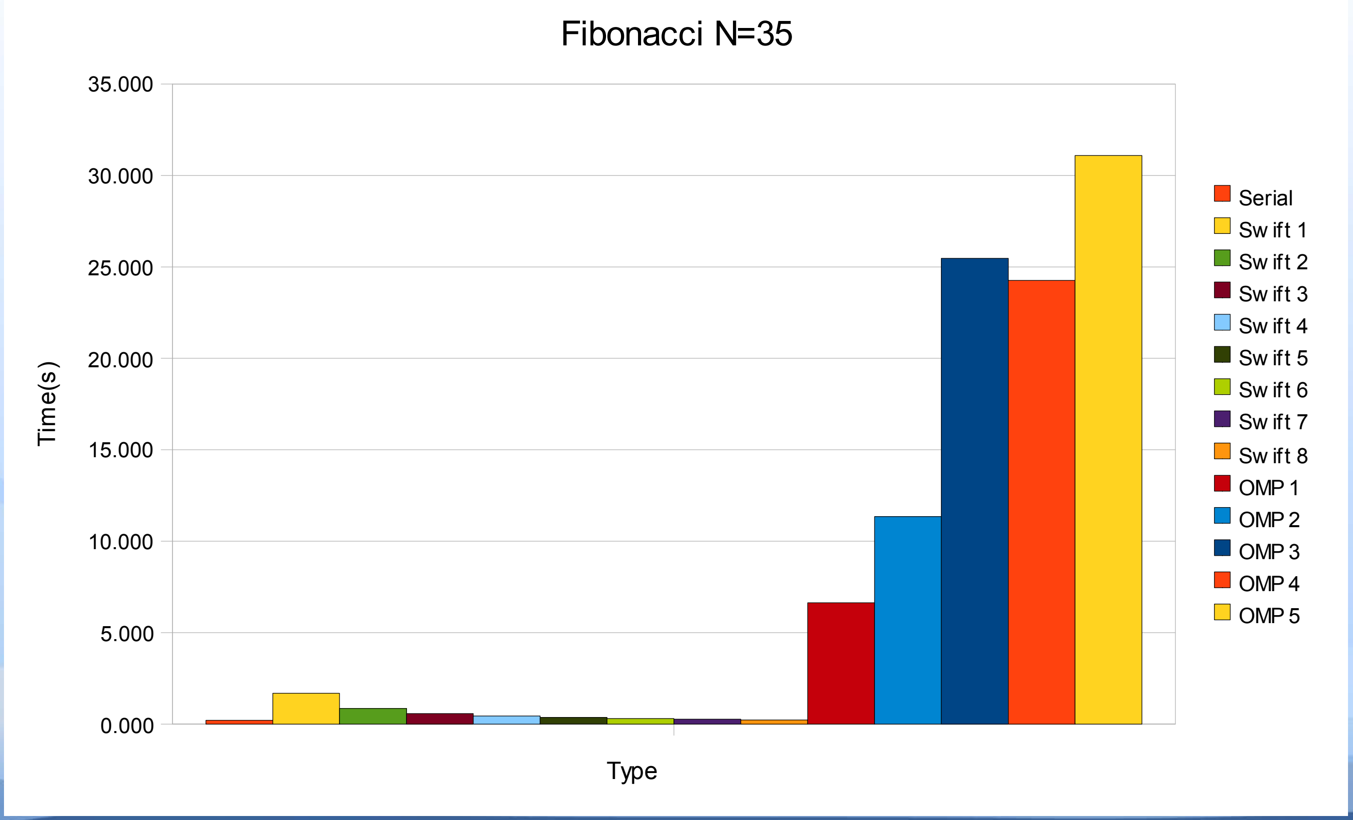
```
typedef struct BZ2_compressStart_data {
        FILE *in;
        FILE *out;
        int blockSize100k;
        int verbosity;
        int workFactor;
        int *r;
        // sync related
        swift_size_t sync_frames_remaining;
        char _pad[SWIFT_CACHE_LINE_SIZE - sizeof(swift_size_t)];
} BZ2_compressStart_data_t;

typedef struct BZ2_compressBlockTask_data {
        EState *s;
        hyper_writer *output;
        int *r;
        char _pad[SWIFT_CACHE_LINE_SIZE - sizeof(int)];
} BZ2_compressBlockTask_data_t;

void
BZ2_compressBlockTask (swift_thread_t *thread, swift_frame_t *frame)
{
        // BZ2_compressBlockCilk(EState *s, hyper_writer &output)
        BZ2_compressBlockTask_data_t *data =
             (BZ2_compressBlockTask_data_t *) frame->private_data;
        swift_status_t status;

        EState *s = data->s;
        hyper_writer output = *data->output;

        SWIFT_LOG_FRAME_INFO_STR("\nBZ2_compressBlockTask() ", thread, frame);

        BZ2_compressBlock(s, output);

        swift_signal_frame_done(thread, frame, &status);
}
```

4. Test results

Intel Pentium Xeon

```
$ cat /proc/cpuinfo |egrep 'name|MHz|cache size|bogo'
model name      : Intel(R) Xeon(R) CPU           E5405  @ 2.00GHz
cpu MHz         : 2000.117
cache size      : 6144 KB
Bogomips        : 4000.23
   ... 7 more like this ...
```

-O2
gcc 4.4.0



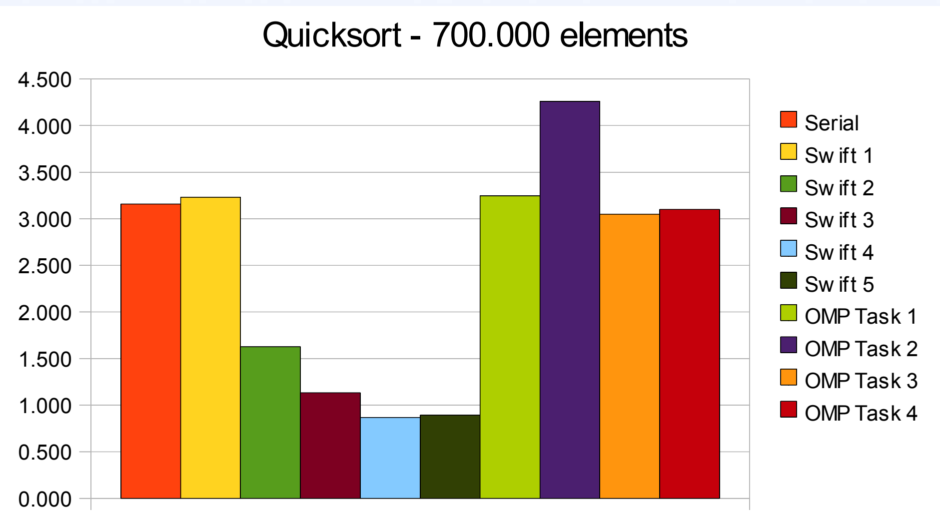

```
int  fibo(int n)
{
    int x, y;

    if (n < 2) {
        return n;
    }


    #pragma omp task shared(x)
    x = fibo(n - 1);

    #pragma omp task shared(y)
    y = fibo(n - 2);

    #pragma omp taskwait
    return x + y;
}
```

Type	Time (s)
Serial	0.21
Swift 1	1.680
Swift 2	0.850
Swift 3	0.570
Swift 4	0.440
Swift 5	0.360
Swift 6	0.300
Swift 7	0.260
Swift 8	0.220
OMP 1	6.628
OMP 2	11.348
OMP 3	25.471
OMP 4	24.266
OMP 5	31.091

4. Test results

Intel Pentium Xeon

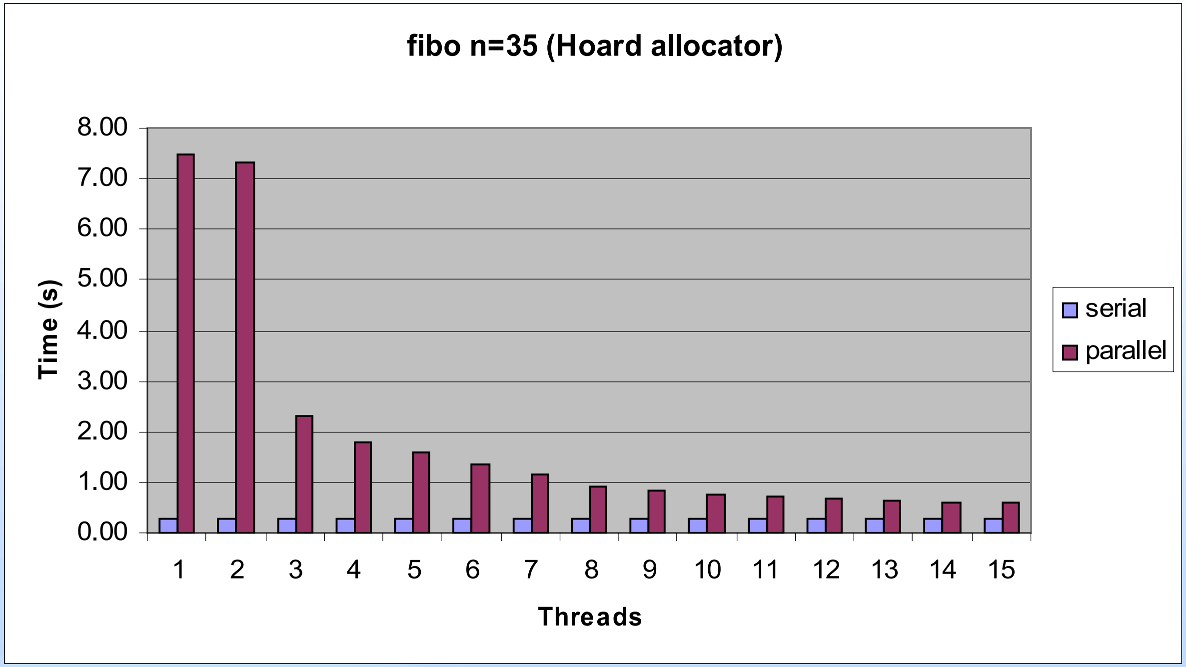
```
$ cat /proc/cpuinfo |egrep 'name|MHz|cache size|bogo'
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
cpu MHz         : 2400.136
cache size      : 4096 KB
bogomips        : 4800.27
   ... 3 more like this ...

-O2
gcc 4.4.1
```



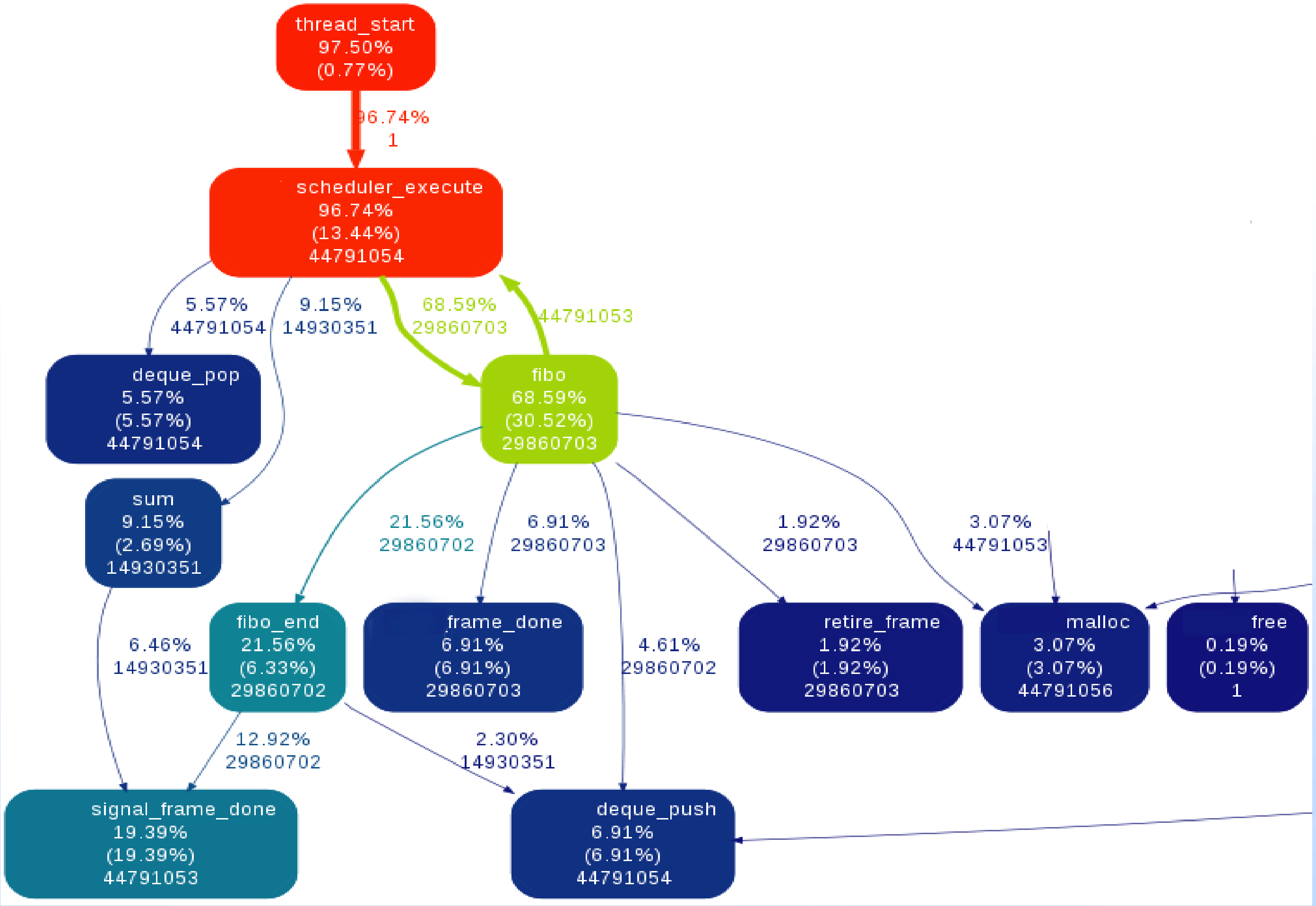

```
quicksort_omp_par (int *data, int p, int r)
{
  if (p < r) {
     int q = partition (data, p, r);
     #pragma omp parallel sections firstprivate(data, p, q, r)
     {
       #pragma omp section
       quicksort_omp_par(data, p, q-1);
       #pragma omp section
       quicksort_omp_par(data, q+1, r);
     }
  }
}

---------------------------------------------

quicksort_omp_par (int *data, int p, int r)
{
  if (p < r) {
    int q = partition (data, p, r);

     #pragma omp task
     quicksort_omp_par(data, p, q-1);

     #pragma omp task
     quicksort_omp_par(data, q+1, r);

     #pragma omp taskwait
  }
}
```

4. Test results

Intel Pentium 16 cores



4. Test results



Conclusions

- **efficient scheduler on parallel platforms**
- **(very close to) optimal speedup**
- **a larger class of problems (vs Cilk)**

Thank you!