CIS 431/531
Intro to Parallel Computing

OpenMP III

Nested Parallelism

OpenMP parallel regions can be nested inside each other.

- If nested parallelism is disabled, then the new team created by a thread encountering a parallel construct inside a parallel region consists only of the encountering thread (i.e., one thread).
- If nested parallelism is enabled, then the new team may consist of more than one thread.

Nested Parallelism - Example

```
void report_num_threads(int level)
{
    #pragma omp single
    {
        printf("Level %d: number of threads in the team - %d\n",
                  level, omp_get_num_threads());
    }
 }


    omp_set_dynamic(0);
    #pragma omp parallel num_threads(2) // create thread a, b
    {
        report_num_threads(1);
        #pragma omp parallel num_threads(2) // create thread c, d
        {
            report_num_threads(2);
            #pragma omp parallel num_threads(2)// create thread e, f
            {
                report_num_threads(3);
            }
        }
    }
```

Nested Parallelism - Example

```
export OMP_NESTED=TRUE
./a.out
# thread a and b -> omp single -> 1 print
Level 1: number of threads in the team - 2

# thread a reaches nested parallel, create a.c and a.d
# thread b reaches nested parallel, create b.c and b.d
# thread a.c and a.d -> omp single -> 1 print
Level 2: number of threads in the team - 2
# thread b.c and b.d -> omp single -> 1 print
Level 2: number of threads in the team - 2

# thread a.c reaches nested parallel, create a.c.e and a.c.f
# thread a.d reaches nested parallel, create a.d.e and a.d.f
# thread a.c.e and a.c.f -> omp single -> 1 print
Level 3: number of threads in the team - 2
# thread a.d.e and a.d.f -> omp single -> 1 print
Level 3: number of threads in the team - 2

# thread b.c reaches nested parallel, create b.c.e and b.c.f
# thread b.d reaches nested parallel, create b.d.e and b.d.f
# thread b.c.e and b.c.f -> omp single -> 1 print
Level 3: number of threads in the team - 2
# thread b.d.e and b.d.f -> omp single -> 1 print
Level 3: number of threads in the team - 2
```

Nested Parallelism - Example

```
export OMP_NESTED=FALSE
./a.out
# thread a and b -> omp single -> 1 print
Level 1: number of threads in the team - 2

# thread a reaches nested parallel, cannot create new threads (only a exists)
# thread a prints
Level 2: number of threads in the team - 1

# thread a reaches nested parallel, cannot create new threads (only a exists)
# thread a prints
Level 3: number of threads in the team - 1

# thread b reaches nested parallel, cannot create new threads (only b exists)
# thread b prints
Level 2: number of threads in the team - 1

# thread b reaches nested parallel, cannot create new threads (only b exists)
# thread b prints
Level 3: number of threads in the team - 1
```

Nested Parallelism

`omp_set_num_threads()`

- Sets the number of OpenMP threads available for use in ***subsequent*** parallel regions.

`omp_get_num_threads()`

- Number of threads in the current parallel region.

`omp_get_max_threads()`

- returns the ***maximum*** available threads for use in *subsequent* parallel region.

`omp_set_nested()`

- same as `OMP_NESTED=TRUE/FALSE`

`omp_get_nested()`

- Returns true if nested parallel regions are enabled.

Nested Parallelism

```
omp_set_nested(1);
omp_set_dynamic(0);
#pragma omp parallel num_threads(2)
{
    if (omp_get_thread_num() == 0)
        omp_set_num_threads(4);
    else
        omp_set_num_threads(6);

    printf("%d: %d %d\n",
           omp_get_thread_num(),
           omp_get_num_threads(),
           omp_get_max_threads());

    #pragma omp parallel
    {
        #pragma omp master
        {
            printf("Inner: %d\n",
              omp_get_num_threads());
        }
        omp_set_num_threads(7);
    }

    #pragma omp parallel
    {
        printf("count me.\n");
    }
}
```

Nested Parallelism

```
0: 2 4
1: 2 6
Inner: 4
count me.
Inner: 6
count me.
count me.
count me.
count me.
count me.
count me.
count me.
count me.
count me.
count me.
```

Nested Parallelism

Why?

- Generating threads in a nested fashion may map naturally to the parallel workload (i.e., makes it easier to understand/code)
- Not enough work in the outermost parallel region - better utilization of resources

Be careful not to oversubscribe threads to cores - creating, managing, and synchronizing threads is not cheap.

Questions?

Dynamic Threading

Dynamic threading (`omp_set_dynamic`)

- A value that indicates if the number of threads available in upcoming parallel regions can be adjusted by the OpenMP runtime. If nonzero, the runtime can adjust the number of threads, if zero, the runtime won't dynamically adjust the number of threads.

Why?

- Having many threads means higher management and synchronization costs.
- In some cases, as you reach the end of your workload, having fewer threads makes sense - using dynamic threading lets the runtime do this.

Thread Binding

Threads are software, cores are hardware.

The OS can move threads between cores: **not** a good idea for performance.

Set "export OMP_PROC_BIND=true" and you'll be good in most cases.

You can also specify which threads goes to which core manually - this is called ***affinity***.

Thread Affinity

To understand binding, you also need to understand how the hardware is "laid out."

Look at `/proc/cpuinfo` and `lscpu`

```
[jeec@talapas-ln1 ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    2
Core(s) per socket:    14
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
Stepping:              1
CPU MHz:               3346.166
CPU max MHz:           3500.0000
CPU min MHz:           1200.0000
BogoMIPS:              5200.57
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              35840K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55
```

Thread Affinity

To understand binding, you also need to understand how the hardware is "laid out."

Look at `/proc/cpuinfo` and `lscpu`

```
Processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
stepping        : 1
microcode       : 0xb00003e
cpu MHz         : 2734.411
cache size      : 35840 KB
physical id     : 0
siblings        : 28
core id         : 0
cpu cores       : 14
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes

...

processor       : 1
vendor_id       : GenuineIntel

...
```

Thread Affinity

To understand binding, you also need to understand how the hardware is "laid out."

Look at `/proc/cpuinfo` and `lscpu`

- `physical id` - physical CPU (socket)
- `core id` - physical core within a CPU
- `processor` - hardware thread

Thread Affinity - Example

```
lscpu | grep -i 'core\|thread\|socket'
```

Thread(s) per core: 2

Core(s) per socket: 14

Socket(s): 2

```
cat /proc/cpuinfo
```

```
processor       : 55
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
stepping        : 1
microcode       : 0xb000038
cpu MHz         : 2614.599
cache size      : 35840 KB
physical id     : 1
siblings        : 28
core id         : 14
cpu cores       : 14
apicid          : 61
initial apicid  : 61
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3
sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c
rdrand lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3 intel_pt ssbd ibrs ibpb stibp tpr_shadow vnmi
flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a rdseed adx
xsaveopt cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts md_clear spec_ctrl
intel_stibp flush_l1d
bogomips        : 5203.53
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Thread Affinity - Example

```
awk '/processor|core id|physical id/ {arr[j++]=$0};
 END{for(i=0;i<j;i+=3) {printf "%-20s %-20s %-20s\n",
 arr[i+1],arr[i+2],arr[i]} }' /proc/cpuinfo |  sed -e 's/[ \t]/ /g' | sort -n -k4,4 -k8,8 -k11,11
physical id : 0      core id  : 0        processor : 0
physical id : 0      core id  : 0        processor : 28
physical id : 0      core id  : 1        processor : 2
physical id : 0      core id  : 1        processor : 30
physical id : 0      core id  : 2        processor : 4
physical id : 0      core id  : 2        processor : 32
physical id : 0      core id  : 3        processor : 6
physical id : 0      core id  : 3        processor : 34
physical id : 0      core id  : 4        processor : 8
physical id : 0      core id  : 4        processor : 36
physical id : 0      core id  : 5        processor : 10
physical id : 0      core id  : 5        processor : 38
physical id : 0      core id  : 6        processor : 12
physical id : 0      core id  : 6        processor : 40
physical id : 0      core id  : 8        processor : 14
physical id : 0      core id  : 8        processor : 42
…
physical id : 0      core id  : 14       processor : 26
physical id : 0      core id  : 14       processor : 54
physical id : 1      core id  : 0        processor : 1
physical id : 1      core id  : 0        processor : 29
physical id : 1      core id  : 1        processor : 3
physical id : 1      core id  : 1        processor : 31
physical id : 1      core id  : 2        processor : 5
physical id : 1      core id  : 2        processor : 33
physical id : 1      core id  : 3        processor : 7
physical id : 1      core id  : 3        processor : 35
physical id : 1      core id  : 4        processor : 9
physical id : 1      core id  : 4        processor : 37
physical id : 1      core id  : 5        processor : 11
physical id : 1      core id  : 5        processor : 39
physical id : 1      core id  : 6        processor : 13
physical id : 1      core id  : 6        processor : 41
physical id : 1      core id  : 8        processor : 15
physical id : 1      core id  : 8        processor : 43
...
physical id : 1      core id  : 14       processor : 27
physical id : 1      core id  : 14       processor : 55
```

Thread Affinity

`OMP_PROC_BIND` – describes how threads are bound to places

- `false` – no binding
- `true` – lock threads to a core
- `master` – allocate threads in the same place as the master thread
- `close` – place threads close to the master thread
- `spread` – spread out the threads as much as possible (evenly)

`OMP_PLACES` – describes these places in terms of the available hardware

`OMP DISPLAY ENV` - displays the bindings at runtime (good for making sure it's doing what you think it's doing).

On Intel systems, `KMP_AFFINITY='verbose'` displays thread to place mapping

Thread Affinity - Example

Default

```
OPENMP DISPLAY ENVIRONMENT BEGIN
   _OPENMP='201511'
  [host] OMP_CANCELLATION='FALSE'
  [host] OMP_DEFAULT_DEVICE='0'
  [host] OMP_DISPLAY_ENV='TRUE'
  [host] OMP_DYNAMIC='FALSE'
  [host] OMP_MAX_ACTIVE_LEVELS='2147483647'
  [host] OMP_MAX_TASK_PRIORITY='0'
  [host] OMP_NESTED='FALSE'
  [host] OMP_NUM_THREADS: value is not defined
  [host] OMP_PLACES: value is not defined
  [host] OMP_PROC_BIND='false'
  [host] OMP_SCHEDULE='static'
  [host] OMP_STACKSIZE='4M'
  [host] OMP_THREAD_LIMIT='2147483647'
  [host] OMP_WAIT_POLICY='PASSIVE'
OPENMP DISPLAY ENVIRONMENT END
```

Thread Affinity - Example

```
OMP_PROC_BIND='true'

OPENMP DISPLAY ENVIRONMENT BEGIN
   _OPENMP='201511'
  [host] OMP_CANCELLATION='FALSE'
  [host] OMP_DEFAULT_DEVICE='0'
  [host] OMP_DISPLAY_ENV='TRUE'
  [host] OMP_DYNAMIC='FALSE'
  [host] OMP_MAX_ACTIVE_LEVELS='2147483647'
  [host] OMP_MAX_TASK_PRIORITY='0'
  [host] OMP_NESTED='FALSE'
  [host] OMP_NUM_THREADS: value is not defined
  [host] OMP_PLACES='cores'
  [host] OMP_PROC_BIND='spread'
  [host] OMP_SCHEDULE='static'
  [host] OMP_STACKSIZE='4M'
  [host] OMP_THREAD_LIMIT='2147483647'
  [host] OMP_WAIT_POLICY='PASSIVE'
OPENMP DISPLAY ENVIRONMENT END
```

Thread Affinity - Example

```
export OMP_NUM_THREADS=8
   [host] OMP_PLACES='cores'
   [host] OMP_PROC_BIND='spread'
```

```
OMP: Info #247: KMP_AFFINITY: pid 191703 tid 191703 thread 0 bound to OS proc set {0,28}
My TID is 0
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191707 thread 4 bound to OS proc set {1,29}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191706 thread 3 bound to OS proc set {22,50}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191708 thread 5 bound to OS proc set {9,37}
My TID is 4
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191710 thread 7 bound to OS proc set {23,51}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191705 thread 2 bound to OS proc set {14,42}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191709 thread 6 bound to OS proc set {15,43}
My TID is 3
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191704 thread 1 bound to OS proc set {8,36}
My TID is 5
My TID is 2
My TID is 1
My TID is 7
My TID is 6
```

Thread Affinity - Example

```
physical id : 0     core id  : 0     processor : 0
physical id : 0     core id  : 0     processor : 28     Thread 0
physical id : 0     core id  : 1     processor : 2
physical id : 0     core id  : 1     processor : 30
physical id : 0     core id  : 2     processor : 4
physical id : 0     core id  : 2     processor : 32
physical id : 0     core id  : 3     processor : 6
physical id : 0     core id  : 3     processor : 34
physical id : 0     core id  : 4     processor : 8
physical id : 0     core id  : 4     processor : 36     Thread 1
physical id : 0     core id  : 5     processor : 10
physical id : 0     core id  : 5     processor : 38
physical id : 0     core id  : 6     processor : 12
physical id : 0     core id  : 6     processor : 40
physical id : 0     core id  : 8     processor : 14
physical id : 0     core id  : 8     processor : 42     Thread 2
physical id : 0     core id  : 9     processor : 16
physical id : 0     core id  : 9     processor : 44
physical id : 0     core id  : 10    processor : 18
physical id : 0     core id  : 10    processor : 46
physical id : 0     core id  : 11    processor : 20
physical id : 0     core id  : 11    processor : 48
physical id : 0     core id  : 12    processor : 22
physical id : 0     core id  : 12    processor : 50     Thread 3
physical id : 0     core id  : 13    processor : 24
physical id : 0     core id  : 13    processor : 52
physical id : 0     core id  : 14    processor : 26
physical id : 0     core id  : 14    processor : 54
```

Thread Affinity - Example

```
export OMP_NUM_THREADS=8
   [host] OMP_PLACES='cores'
   [host] OMP_PROC_BIND='spread'
```

```
OMP: Info #247: KMP_AFFINITY: pid 191703 tid 191703 thread 0 bound to OS proc set {0,28}
My TID is 0
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191707 thread 4 bound to OS proc set {1,29}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191706 thread 3 bound to OS proc set {22,50}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191708 thread 5 bound to OS proc set {9,37}
My TID is 4
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191710 thread 7 bound to OS proc set {23,51}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191705 thread 2 bound to OS proc set {14,42}
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191709 thread 6 bound to OS proc set {15,43}
My TID is 3
OMP: Info #247: OMP_PROC_BIND: pid 191703 tid 191704 thread 1 bound to OS proc set {8,36}
My TID is 5
My TID is 2
My TID is 1
My TID is 7
My TID is 6
```

Thread Affinity - Example

```
physical id : 1      core id  : 0        processor : 1
physical id : 1      core id  : 0        processor : 29     Thread 4
physical id : 1      core id  : 1        processor : 3
physical id : 1      core id  : 1        processor : 31
physical id : 1      core id  : 2        processor : 5
physical id : 1      core id  : 2        processor : 33
physical id : 1      core id  : 3        processor : 7
physical id : 1      core id  : 3        processor : 35
physical id : 1      core id  : 4        processor : 9
physical id : 1      core id  : 4        processor : 37     Thread 5
physical id : 1      core id  : 5        processor : 11
physical id : 1      core id  : 5        processor : 39
physical id : 1      core id  : 6        processor : 13
physical id : 1      core id  : 6        processor : 41
physical id : 1      core id  : 8        processor : 15
physical id : 1      core id  : 8        processor : 43     Thread 6
physical id : 1      core id  : 9        processor : 17
physical id : 1      core id  : 9        processor : 45
physical id : 1      core id  : 10       processor : 19
physical id : 1      core id  : 10       processor : 47
physical id : 1      core id  : 11       processor : 21
physical id : 1      core id  : 11       processor : 49
physical id : 1      core id  : 12       processor : 23
physical id : 1      core id  : 12       processor : 51     Thread 7
physical id : 1      core id  : 13       processor : 25
physical id : 1      core id  : 13       processor : 53
physical id : 1      core id  : 14       processor : 27
physical id : 1      core id  : 14       processor : 55
```

Thread Affinity - Example

```
export OMP_PROC_BIND='master'
  [host] OMP_PLACES='cores'
  [host] OMP_PROC_BIND='master'
```

```
OMP: Info #247: KMP_AFFINITY: pid 195405 tid 195405 thread 0 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195406 thread 1 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195407 thread 2 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195408 thread 3 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195409 thread 4 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195410 thread 5 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195411 thread 6 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 195405 tid 195412 thread 7 bound to OS proc set {0,28}
```

All threads have been assigned to the first core

Thread Affinity - Example

```
export OMP_PROC_BIND=’close’
  [host] OMP_PLACES='cores'
  [host] OMP_PROC_BIND=’close’
```

```
OMP: Info #247: KMP_AFFINITY: pid 428 tid 428 thread 0 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 429 thread 1 bound to OS proc set {2,30}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 430 thread 2 bound to OS proc set {4,32}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 431 thread 3 bound to OS proc set {6,34}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 432 thread 4 bound to OS proc set {8,36}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 433 thread 5 bound to OS proc set {10,38}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 434 thread 6 bound to OS proc set {12,40}
OMP: Info #247: OMP_PROC_BIND: pid 428 tid 435 thread 7 bound to OS proc set {14,42}
```

All threads have been assigned to the cores 0-7 of socket 0.

Thread Affinity - Example

```
[host] OMP_NUM_THREADS='56'
[host] OMP_PLACES='cores'
[host] OMP_PROC_BIND='close'

OMP: Info #247: KMP_AFFINITY: pid 3287 tid 3287 thread 0 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3289 thread 1 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3290 thread 2 bound to OS proc set {2,30}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3291 thread 3 bound to OS proc set {2,30}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3292 thread 4 bound to OS proc set {4,32}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3293 thread 5 bound to OS proc set {4,32}
…
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3341 thread 50 bound to OS proc set
{23,51}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3342 thread 51 bound to OS proc set
{23,51}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3343 thread 52 bound to OS proc set
{25,53}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3344 thread 53 bound to OS proc set
{25,53}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3345 thread 54 bound to OS proc set
{27,55}
OMP: Info #247: OMP_PROC_BIND: pid 3287 tid 3346 thread 55 bound to OS proc set
{27,55}
```

Linear placing of threads to processors

Thread Affinity - Example

```
[host] OMP_NUM_THREADS='56'
[host] OMP_PLACES='cores'
[host] OMP_PROC_BIND='spread'

OMP: Info #247: KMP_AFFINITY: pid 5077 tid 5077 thread 0 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5079 thread 1 bound to OS proc set {0,28}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5080 thread 2 bound to OS proc set {2,30}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5081 thread 3 bound to OS proc set {2,30}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5082 thread 4 bound to OS proc set {4,32}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5083 thread 5 bound to OS proc set {4,32}
…
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5129 thread 51 bound to OS proc set
{23,51}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5130 thread 52 bound to OS proc set
{25,53}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5131 thread 53 bound to OS proc set
{25,53}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5133 thread 54 bound to OS proc set
{27,55}
OMP: Info #247: OMP_PROC_BIND: pid 5077 tid 5134 thread 55 bound to OS proc set
{27,55}
```

No difference

Thread Affinity

`OMP_PLACES` – describes these places in terms of the available hardware

- `threads` – each place corresponds to a hardware thread.
- `cores` – each place corresponds to a core
- `sockets` – each place corresponds to a socket
- A list with explicit place values (for process IDs)
  - A place: {0}
  - Place List: {0},{1},{2},{3}
  - Interval notation: <place>:<len>:<stride>

Thread Affinity - Examples

OMP_NUM_THREADS=8; OMP_PLACES=threads

OMP_NUM_THREADS=8; OMP_PLACES=cores

Thread Affinity - Examples

```
[host] OMP_NUM_THREADS='8'
[host] OMP_PLACES='{0}:7:1'
   -> ‘places’ start from 0; stride 1; 7 ‘places’
   -> 0, 1, 2, 3, 4, 5, 6
   -> with 8 threads, you don’t have enough places for a unique one for each, so thread 0 and 1 are assigned to h/w thread 0
```

```
OMP: Info #247: KMP_AFFINITY: pid 8313 tid 8313 thread 0 bound to OS proc set {0}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8316 thread 3 bound to OS proc set {2}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8317 thread 4 bound to OS proc set {3}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8318 thread 5 bound to OS proc set {4}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8314 thread 1 bound to OS proc set {0}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8319 thread 6 bound to OS proc set {5}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8315 thread 2 bound to OS proc set {1}
OMP: Info #247: OMP_PROC_BIND: pid 8313 tid 8320 thread 7 bound to OS proc set {6}
```

Thread Affinity - Examples

```
[host] OMP_NUM_THREADS='8'
[host] OMP_PLACES='{0}:8:1'
```

```
OMP: Info #247: KMP_AFFINITY: pid 86654 tid 86654 thread 0 bound to OS proc set {0}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86656 thread 2 bound to OS proc set {2}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86657 thread 3 bound to OS proc set {3}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86660 thread 6 bound to OS proc set {6}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86659 thread 5 bound to OS proc set {5}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86661 thread 7 bound to OS proc set {7}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86655 thread 1 bound to OS proc set {1}
OMP: Info #247: OMP_PROC_BIND: pid 86654 tid 86658 thread 4 bound to OS proc set {4}
```

Questions?

First Touch

Remember that

- Memory allocated with *malloc* and other similar routines is not actually allocated physically
- Actual allocation only happens when data is written to it (e.g., initialization)
- In light of this, consider the following OpenMP code:

```
double *x = (double*) malloc(N*sizeof(double));

for (i=0; i<N; i++) {
  x[i] = 0;
}

#pragma omp parallel for

for (i=0; i<N; i++) {
  do something with x[i]

}
```

Problems? (Remember that this system is a NUMA system)

First Touch

Problems? (Remember that this system is a NUMA system)

- Socket where the master thread is located is where all the data is
- Subsequent read from a different socket will required the data to traverse UPI (longer latency & lower bandwidth)

Solution?

- Initialize the data where it will eventually be read

```
#pragma omp parallel for schedule(static)

for (i=0; i<N; i++) {
  x[i] = 0;
}
```

Questions?

Example - Fibonacci

Fibonacci series - a sequence where each term is a sum of the two previous terms

F0 = 0

F1 = 1

F2 = 0 + 1 = 1

F3 = 1 + 1 = 2

F4 = 2 + 1 = 3

F5 = 2 + 3 = 5

F6 = 3 + 5 = 8

F7 = 5 + 8 = 13

Example - Fibonacci

How do we calculate this in parallel?

Typically we can use recursion:

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

Are each recursive call independent?

Example - Fibonacci

How do we calculate this in parallel?

Typically we can use recursion:

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

Are each recursive call independent? Yes - we can use parallel tasks

Each task creates new tasks fib(n-1) and fib(n-2) using nested parallelism

The tasks in the task pool will be executed by OpenMP threads

Example - Fibonacci

Fibonacci series for 35 is 9227465
Time to calculate the Fibonacci series for 35 is: 0.0717677
Fibonacci series for 35 is 9227465
Time to calculate the Fibonacci series for 35 is: 14.3452

Which is the parallel one?

Example - Fibonacci

Fibonacci series for 35 is 9227465
Time to calculate the Fibonacci series for 35 is: 0.0717677
Fibonacci series for 35 is 9227465
Time to calculate the Fibonacci series for 35 is: 14.3452

Parallel is MUCH slower - why?

- The calculation within each task is basically generating a new task and then adding two numbers together - not enough work.
- Cost of creating new threads/tasks is NOT FREE (as is the synchronization).
- Code is taking more time managing threads/tasks.

Questions?

Prefix Sum

Also referred to as cumulative sum, inclusive scan, scan etc.

$$y_0 = x_0$$

$$y_1 = x_0 + x_1$$

$$y_2 = x_0 + x_1 + x_2$$

Input	1	2	3	4	5	6	…
Prefix sum	1	3	6	10	15	21	…

Prefix Sum

```
for(int i = 0; i < n; i++) {
    prefix[i] = 0;
    for(int j = 0; j <= i; j++) {
        prefix[i] += src[j];
    }
}
```

Work?

Prefix Sum

```
for(int i = 0; i < n; i++) {
    prefix[i] = 0;
    for(int j = 0; j <= i; j++) {
        prefix[i] += src[j];
    }
}
```

Work?

$1 + 2 + \ldots + n = n(n+1)/2 = \frac{1}{2}(n^2 + n) \Rightarrow O(n^2)$

Prefix Sum

Also referred to as cumulative sum, inclusive scan, scan etc.

$$y_0 = x_0$$

$$y_1 = x_0 + x_1$$

$$y_2 = x_0 + x_1 + x_2$$

Input	1	2	3	4	5	6	…
Prefix sum	1	3	6	10	15	21	…

Prefix Sum

```
prefix[0] = src[0];
for(int i = 1; i < n; i++) {
    prefix[i] = src[i] + prefix[i - 1];
}
```

Are the loop iterations independent? (i.e., can they be executed in parallel?)

Prefix Sum

```
prefix[0] = src[0];
for(int i = 1; i < n; i++) {
    prefix[i] = src[i] + prefix[i - 1];
}
```

Are the loop iterations independent? (i.e., can they be executed in parallel?)

- No, there is a RAW dependency (`prefix[i] & prefix[i-1]`)

What can you do?

- You need a new algorithm!

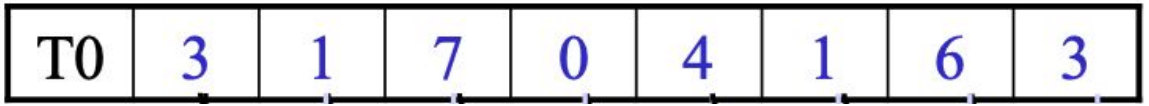
Prefix Sum

Shorter span, more parallel

T0	3	1	7	0	4	1	6	3

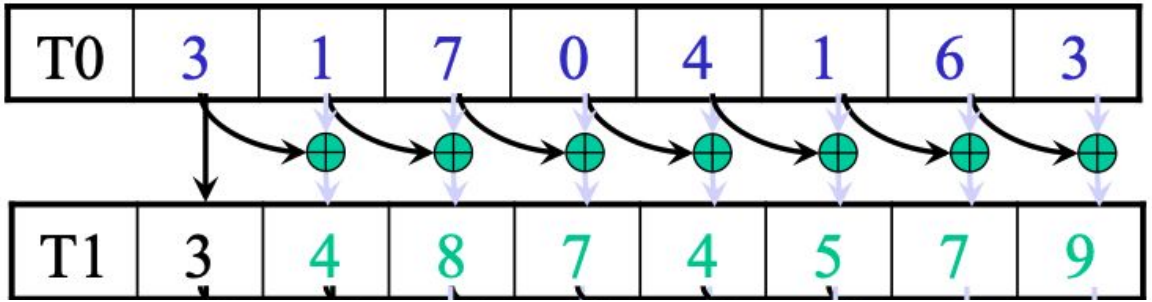
Prefix Sum

Shorter span, more parallel



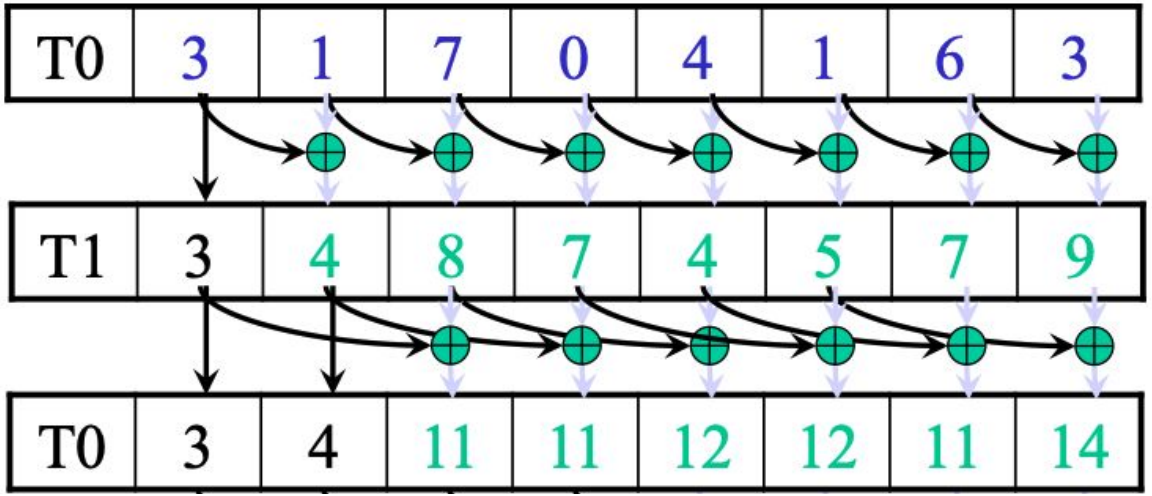
Prefix Sum

Shorter span, more parallel



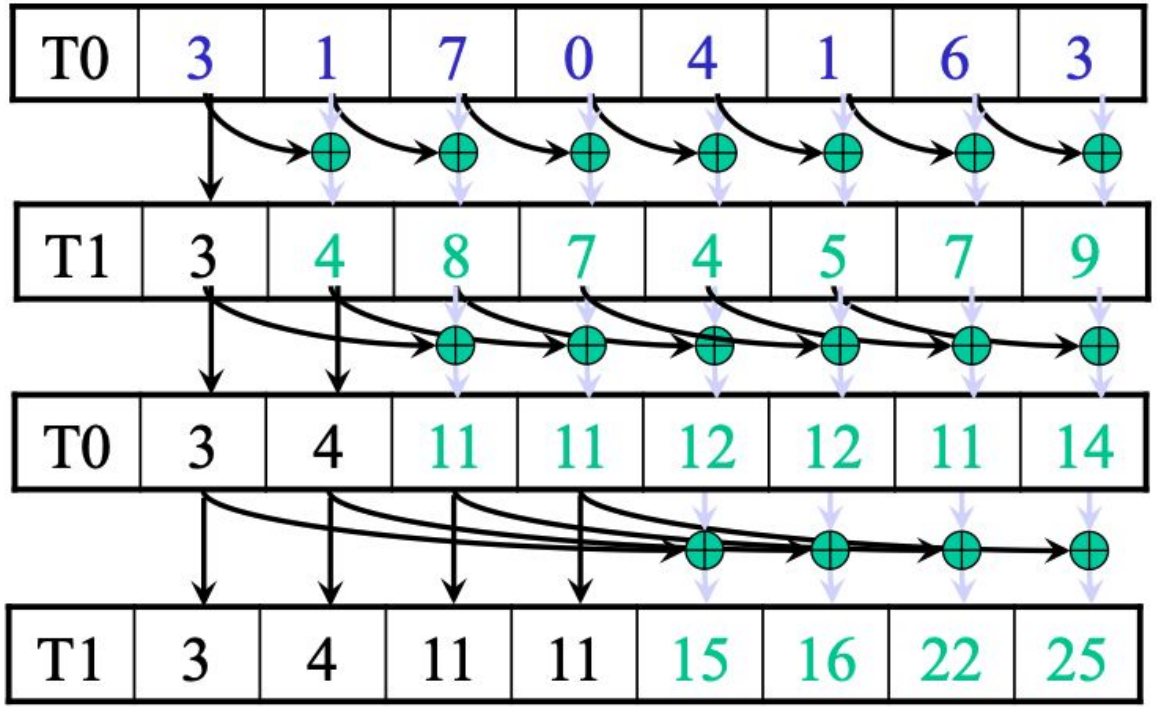
Prefix Sum

Shorter span, more parallel



Prefix Sum

Shorter span, more parallel



Prefix Sum

Shorter span, more parallel

Span:

Parallel:

Work efficiency:

Prefix Sum

Shorter span, more parallel

Span: log(n) iterations

Parallel: Every add in each iteration can be done in parallel

Work efficiency:

Prefix Sum

Shorter span, more parallel

Span: log(n) iterations

Parallel: Every add in each iteration can be done in parallel

Work efficiency:

- We do (n-1), (n-2), (n-4), …, (n - n/2) work
- Total = ?

Serial implementation:

Prefix Sum

Shorter span, more parallel

Span: log(n) iterations

Parallel: Every add in each iteration can be done in parallel

Work efficiency:

We do (n-1), (n-2), (n-4), …, (n - n/2) work

Total = n * log(n) - (n-1) => O(n log(n)) work

Serial implementation:

Prefix Sum

Shorter span, more parallel

Span: log(n) iterations

Parallel: Every add in each iteration can be done in parallel

Work efficiency:

- We do (n-1), (n-2), (n-4), …, (n - n/2) work
- Total = n * log(n) - (n-1) => O(n log(n)) work

Serial implementation:

- Work = n - 1 => O(n) work

Can we do better?

Prefix Sum

Balanced (Binary) Tree

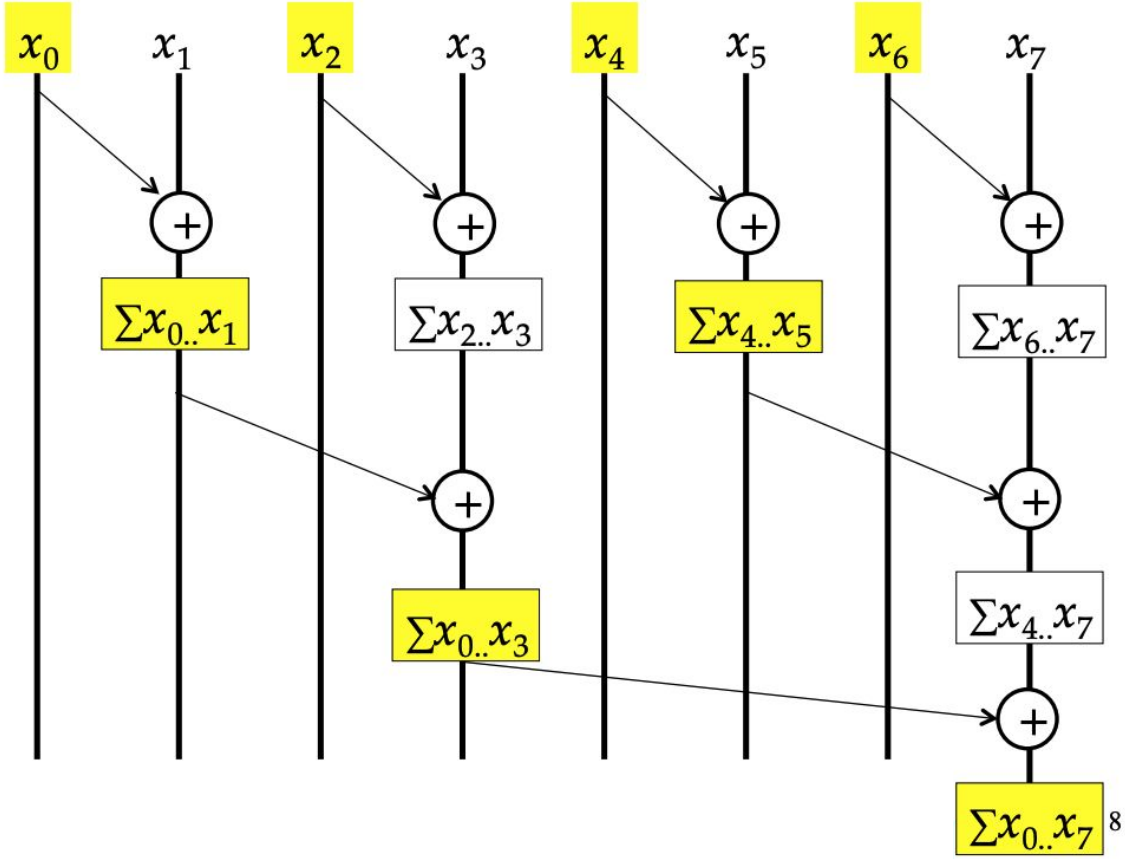
- Build a binary tree on the list
- Traverse the tree to calculate the prefix sum

Traverse the tree up (from leaves to root) and build partial sum at internal nodes of the tree

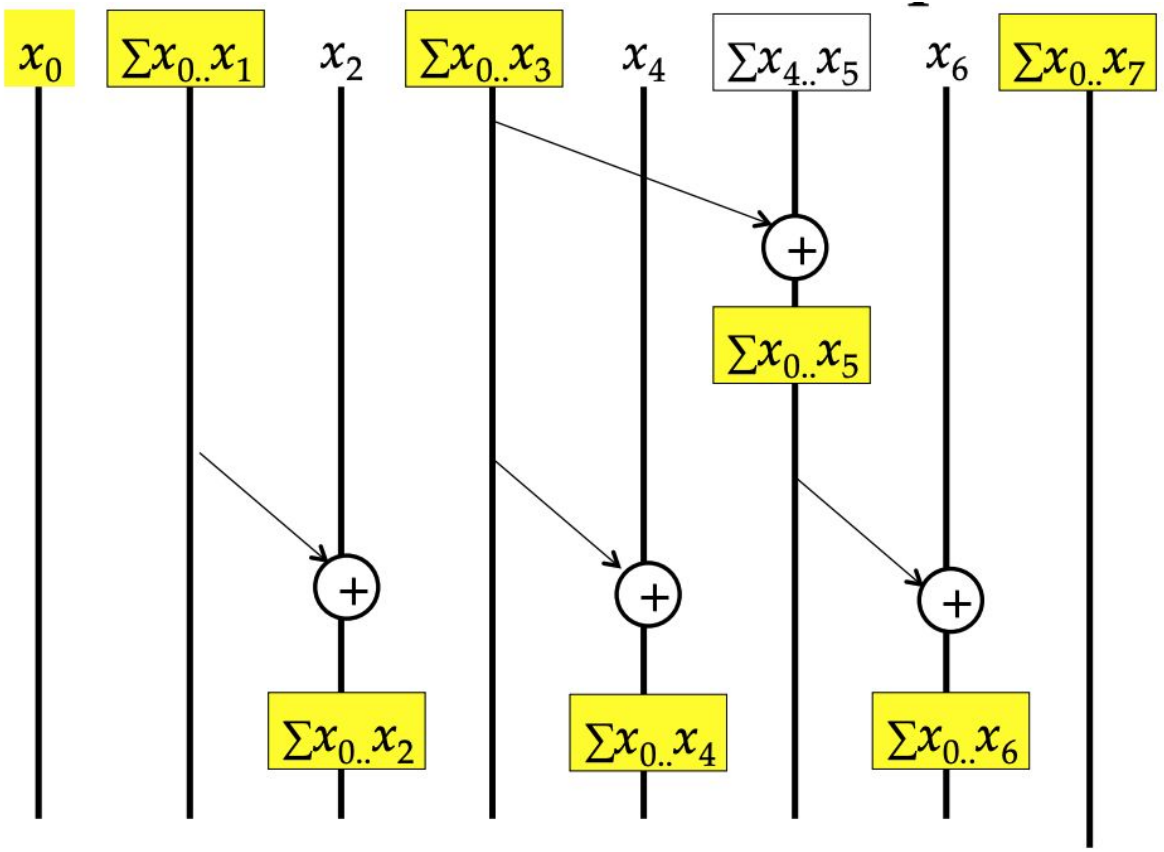
- Root node holds the sum of all leaves

Traverse back down to calculate the prefix sum from the partial sums

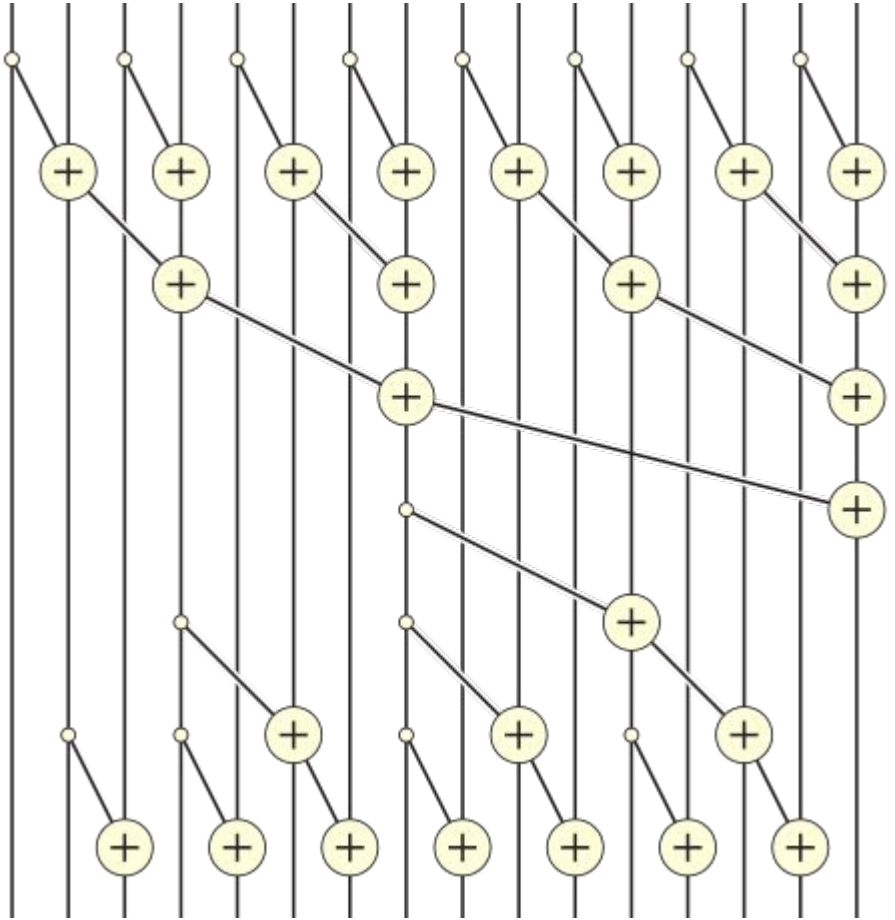
Prefix Sum


x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
$\sum x_{0..}x_1$
$\sum x_{2..}x_3$
$\sum x_{4..}x_5$
$\sum x_{6..}x_7$
$\sum x_{0..}x_3$
$\sum x_{4..}x_7$
$\sum x_{0..}x_7$

Prefix Sum


x_0
$\sum x_{0..}x_1$
x_2
$\sum x_{0..}x_3$
x_4
$\sum x_{4..}x_5$
x_6
$\sum x_{0..}x_7$
+
$\sum x_{0..}x_5$
+
+
+
$\sum x_{0..}x_2$
$\sum x_{0..}x_4$
$\sum x_{0..}x_6$

Prefix Sum



Prefix Sum

Work?

Prefix Sum

Work?

- log(n) + (log(n) - 1) iterations
- (n - 1) + ((n - 1) - log(n)) operations = ~2(n - 1) => O(n) but still 2x more work compared to efficient serial (i.e., (n - 1) version)

However, if we can get speedup using more than 2 cores, you will be better off using the binary tree version.

Homework 2

Implement the two parallel versions of prefix-sum

Example - Fibonacci

```
int calculateFibonacci(int n)
{
    int i;
    int j;
    if(n < 2) {
        return n;
    }
    i = calculateFibonacci(n - 1);
    j = calculateFibonacci(n - 2);
    return (i + j);
}
```

```
[jeec@talapas-ln1 openmp]$ ./a.out 40
Fibonacci series for 40 is 102334155
Time to calculate the Fibonacci series for 40 is: 0.796354
[jeec@talapas-ln1 openmp]$ ./a.out 50
Fibonacci series for 50 is -298632863
Time to calculate the Fibonacci series for 50 is: 99.1724
```

Example - Fibonacci

How would you parallelize this?

```
int calculateFibonacci(int n)
{
    int i;
    int j;
    if(n < 2) {
        return n;
    }
    i = calculateFibonacci(n - 1);
    j = calculateFibonacci(n - 2);
    return (i + j);
}
```

Example - Fibonacci

How would you parallelize this?

```
int calculateFibonacci(int n)
{
    int i;
    int j;
    if(n < 2) {
        return n;
    }
    i = calculateFibonacci(n - 1);
    j = calculateFibonacci(n - 2); /* are these dependent? */
    return (i + j);
}
```

Example - Fibonacci

```
int fib1(int n)
{
    int i;
    int j;
    if (n < 2)
    {
        return n;
    }

    #pragma omp task shared(i) firstprivate(n)
    {
        i = fib1(n - 1);
    }
    #pragma omp task shared(j) firstprivate(n)
    {
        j = fib1(n - 2);
    }
    #pragma omp taskwait
    return (i + j);
}
```

```
#pragma omp parallel shared(n)
{
    #pragma omp single
    {
        printf ("fib(%d) = %d\n", n, fib1(n));
    }
}
```