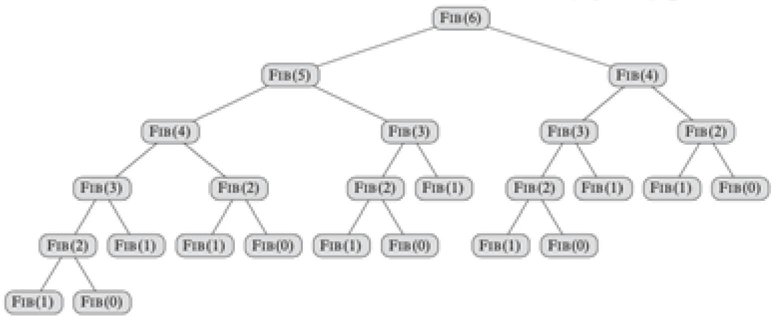
Multi-threaded Algorithms

- Multiprocessors
    - Shared memory multiprocessor model
        - 공통의 메모리를 통해 데이터를 주고 받으면 된다
    - Distributed memory multiprocessor model
- Thread : Serial process
    - Static threading
        - 컴파일 할 때 스레드가 선언됨
    - **Dynamic threading**
        - 런타임에 스레드가 선언됨
- Dynamic multi-threaded programming
    - Parallel loops
    - Nested parallelism
        - Spawn 한 Thread가 또 Spawn할 수 있는 것

Basics of Dynamic Multi-Threading

Computing Fibonacci numbers

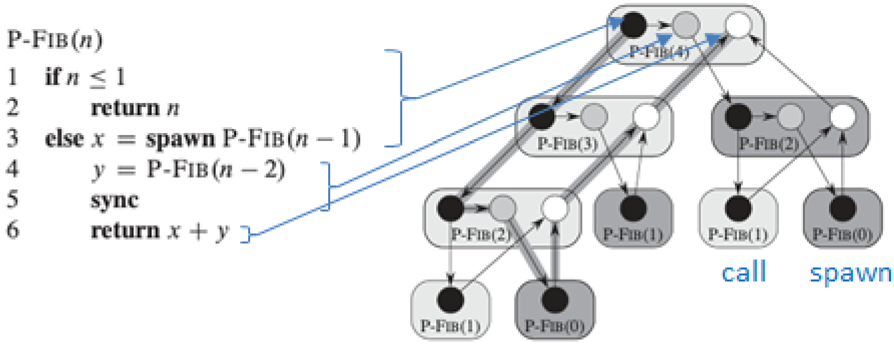
- $$F_0 = 0,$$
  $$F_1 = 1,$$
  $$F_i = F_{i-1} + F_{i-2} \quad \text{for } i \geq 2.$$




```
FIB(n)
if n ≤ 1
    return n
else x = FIB(n − 1)
    y = FIB(n − 2)
    return x + y
```

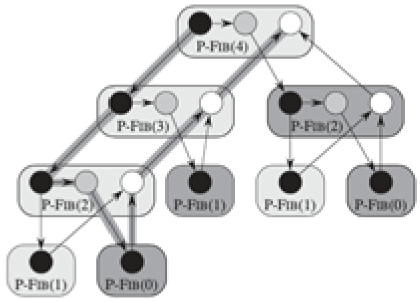
```
P-FIB(n)
if n ≤ 1
    return n
else x = spawn P-FIB(n − 1)
    y = P-FIB(n − 2)
    sync
    return x + y
```

    - spawn : 런타임에 새로운 스레드를 만드는 것
    - sync : 기다리는 것
- Computation DAG(Directed Assigned Graph)



edges represent dependency
- down : spawn/call
- up : return
- horizontal : continuation

- Performance
  - work : 하나의 프로세서를 이용해 계산할 때 걸리는 시간
    = 각 vertex에 걸린 시간의 합 17
  - span : DAG의 임의의 경로를 따라 vertex를 실행할 때 걸리는 가장 긴 시간
    = critical path의 vertex 갯수 8
  - T_p : p 개의 프로세서에 의한 multi-threaded 수행 시간
  - T_1 : sequential execution time 17
  - T_∞ : processor가 충분히 많을 때 수행 시간 8
  - work law : $T_p \geq T_1/P$
  - span law : $T_p \geq T_\infty$



- Speedup
  - work law : $T_p \geq T_1/P$
  - span law : $T_p \geq T_\infty$
  - speedup $= T_1/T_p \leq P$
  - perfect speedup = $T_1/T_p = P$
  - parallelism of the multi-threaded computation = T_1/T_∞

Work: $T_1(A \cup B) = T_1(A) + T_1(B)$
Span: $T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$

(a)

Work: $T_1(A \cup B) = T_1(A) + T_1(B)$
Span: $T_\infty(A \cup B) = \max(T_\infty(A), T_\infty(B))$

(b)

- Parallel for loop

$$y_i = \sum_{j=1}^{n} a_{ij} x_j$$

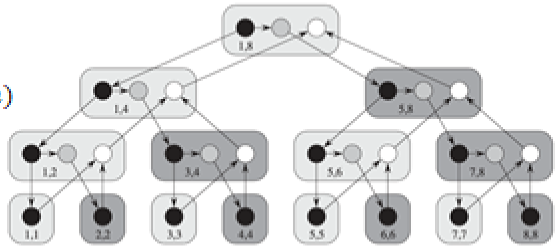
MAT-VEC(A, x)
$n = A.rows$
let y be a new vector of length n
parallel for $i = 1$ **to** n
    $y_i = 0$
parallel for $i = 1$ **to** n
    for $j = 1$ **to** n
        $y_i = y_i + a_{ij} x_j$
return y

MAT-VEC-MAIN-LOOP(A, x, y, n, i, i')
if $i == i'$
    for $j = 1$ **to** n
        $y_i = y_i + a_{ij} x_j$
else $mid = \lfloor (i + i')/2 \rfloor$
    spawn MAT-VEC-MAIN-LOOP(A, x, y, n, i, mid)
    MAT-VEC-MAIN-LOOP$(A, x, y, n, mid + 1, i')$
    sync

$$T_1(n) = \Theta(n^2)$$

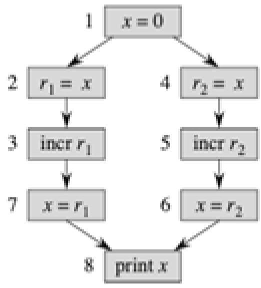
$$T_\infty(n) = \Theta(\lg n) + \max_{1 \le i \le n} iter_\infty(i) = \Theta(n)$$

parallelism = $\Theta(n^2)/\Theta(n) = \Theta(n)$.



- Race conditions

RACE-EXAMPLE( )
$x = 0$
parallel for $i = 1$ **to** 2
    $x = x + 1$
print x



(a)

step	x	r_1	r_2
1	0	–	–
2	0	0	–
3	0	1	–
4	0	1	0
5	0	1	1
6	1	1	1
7	1	1	1

(b)

MAT-VEC-WRONG(A, x)
$n = A.rows$
let y be a new vector of length n
parallel for $i = 1$ **to** n
    $y_i = 0$
parallel for $i = 1$ **to** n
    ~~**parallel for** $j = 1$ **to** n~~
        $y_i = y_i + a_{ij} x_j$
return y

Multi-threaded Matrix Multiplication

P-SQUARE-MATRIX-MULTIPLY(A, B)
$n = A.rows$
let C be a new $n \times n$ matrix
parallel for $i = 1$ **to** n
    parallel for $j = 1$ **to** n
        $c_{ij} = 0$
        for $k = 1$ **to** n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

$$T_1(n) = \Theta(n^3)$$

$$T_\infty(n) = \Theta(n)$$

$$\Theta(n^3)/\Theta(n) = \Theta(n^2).$$

```
P-MATRIX-MULTIPLY-RECURSIVE(C, A, B)
n = A.rows
if n == 1
    c11 = a11 b11
else let T be a new n × n matrix
    partition A, B, C, and T into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22; C11, C12, C21, C22;
        and T11, T12, T21, T22; respectively
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C11, A11, B11)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C12, A11, B12)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C21, A21, B11)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C22, A21, B12)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T11, A12, B21)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T12, A12, B22)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T21, A22, B21)
    P-MATRIX-MULTIPLY-RECURSIVE(T22, A22, B22)
    sync
    parallel for i = 1 to n
        parallel for j = 1 to n
            c_ij = c_ij + t_ij
```

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Then, we can write the matrix product as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix} + \begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}.$$


```
P-MATRIX-MULTIPLY-RECURSIVE(C, A, B)
n = A.rows
if n == 1
    c11 = a11 b11
else let T be a new n × n matrix
    partition A, B, C, and T into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22; C11, C12, C21, C22;
        and T11, T12, T21, T22; respectively
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C11, A11, B11)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C12, A11, B12)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C21, A21, B11)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(C22, A21, B12)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T11, A12, B21)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T12, A12, B22)
    spawn P-MATRIX-MULTIPLY-RECURSIVE(T21, A22, B21)
    P-MATRIX-MULTIPLY-RECURSIVE(T22, A22, B22)
    sync
    parallel for i = 1 to n
        parallel for j = 1 to n
            c_ij = c_ij + t_ij
```


$$M_1(n) = 8M_1(n/2) + \Theta(n^2) = \Theta(n^3)$$

$$M_\infty(n) = M_\infty(n/2) + \Theta(\lg n)$$

$$M_\infty(n) = \Theta(\lg^2 n).$$

$$M_1(n)/M_\infty(n) = \Theta(n^3/\lg^2 n),$$

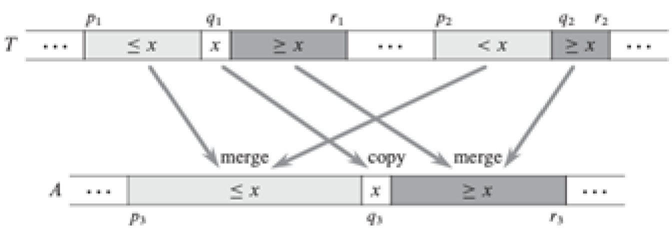
Multi-threaded Merge Sort

```
MERGE-SORT'(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    spawn MERGE-SORT'(A, p, q)
    MERGE-SORT'(A, q + 1, r)
    sync
    MERGE(A, p, q, r)
```

$$MS'_1(n) = 2\,MS'_1(n/2) + \Theta(n) = \Theta(n \lg n),$$

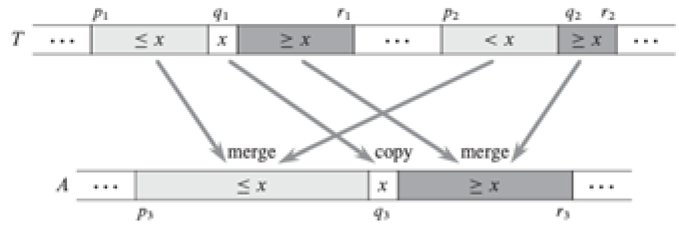
$$MS'_\infty(n) = MS'_\infty(n/2) + \Theta(n) = \Theta(n).$$

$$MS'_1(n)/MS'_\infty(n) = \Theta(\lg n)$$



```
P-MERGE-SORT(A, p, r, B, s)
n = r − p + 1
if n == 1
    B[s] = A[p]
else let T[1..n] be a new array
    q = ⌊(p + r)/2⌋
    q′ = q − p + 1
    spawn P-MERGE-SORT(A, p, q, T, 1)
    P-MERGE-SORT(A, q + 1, r, T, q′ + 1)
    sync
    P-MERGE(T, 1, q′, q′ + 1, n, B, s)
```

```
P-MERGE(T, p1, r1, p2, r2, A, p3)
n1 = r1 − p1 + 1
n2 = r2 − p2 + 1
if n1 < n2                          // ensure that n1 ≥ n2
    exchange p1 with p2
    exchange r1 with r2
    exchange n1 with n2
if n1 == 0                          // both empty?
    return
else q1 = ⌊(p1 + r1)/2⌋
    q2 = BINARY-SEARCH(T[q1], T, p2, r2)
    q3 = p3 + (q1 − p1) + (q2 − p2)
    A[q3] = T[q1]
    spawn P-MERGE(T, p1, q1 − 1, p2, q2 − 1, A, p3)
    P-MERGE(T, q1 + 1, r1, q2, r2, A, q3 + 1)
    sync
```



$$PM_1(n) = \Omega(n) \qquad PM_\infty(n) = \Theta(\lg^2 n) \qquad PM_1(n)/PM_\infty(n) = \Theta(n/\lg^2 n)$$