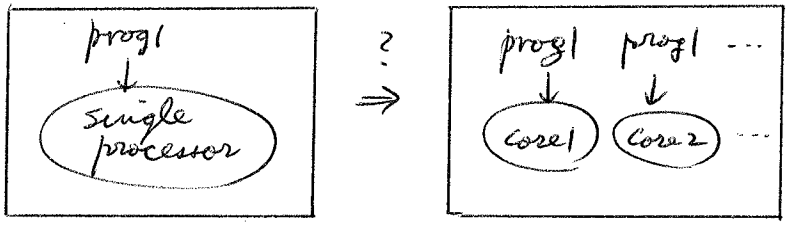
Ch1. Why parallel computing?

- Microprocessor design migration
  - faster single processor — (circuit complexity, high power consumption)
  - ⇒ multicore processor
    - thread-level parallelism (TLP)
- serial (sequential) program → [auto converter] → parallel program
  - (not successful, inefficient)
- HPC application areas
  - climate modeling
  - protein folding
  - drug discovery
  - energy research
  - genomic data analysis
  - various scientific computations
- parallel programs



→ this way cannot make a game program H.P.

parallel version of prog1 on multicore system is needed.

- serial lang. constructs → [auto converter] → parallel constructs
- vs.
- designing a new parallel algo./implementation
  - much better way of achieving HP.

ex) globalSum computation (n values)

— Serial code

```
Sum = ∅;
for (i=∅; i<n; i++)
    { x = compute_next_val(--);
      Sum += x; }
```

[problem size = n
[∃ p processors (cores)

1, 2, 3, 4, ..., n → P1, P2, ..., Pp

— parallel code

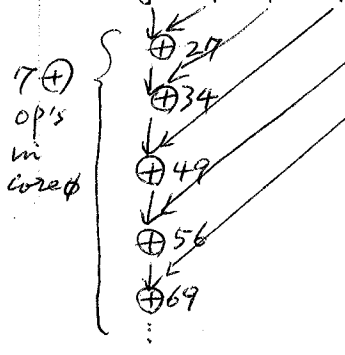
1. compute partial sum in each core:

```
my_sum = ∅; // partial sum
[ my_first_index = ----;
[ my_last_index = ----;
for (my_i = my_first_index; my_i < my_last_index; my_i++)
    { my_x = compute_next_val(--);
      my_sum += my_x; }
```

2. Combine for global_sum: — each core sends partial_sum to the master core

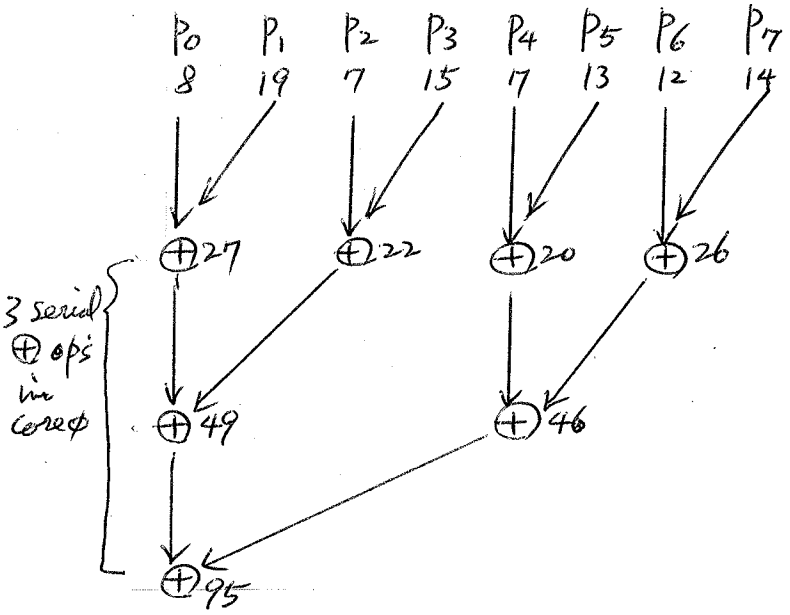
```
if (master_core)
    { sum = my_sum;
      for (each core other than myself)
          { receive partial_sum from core;
            sum += value(partial_sum); }
else
    send my_sum to the master core;
```

ex) P0 P1 P2 P3 P4 P5 P6 P7 — ∃ 8 cores
     8 19 7 15 7 13 12 14 — partial sums



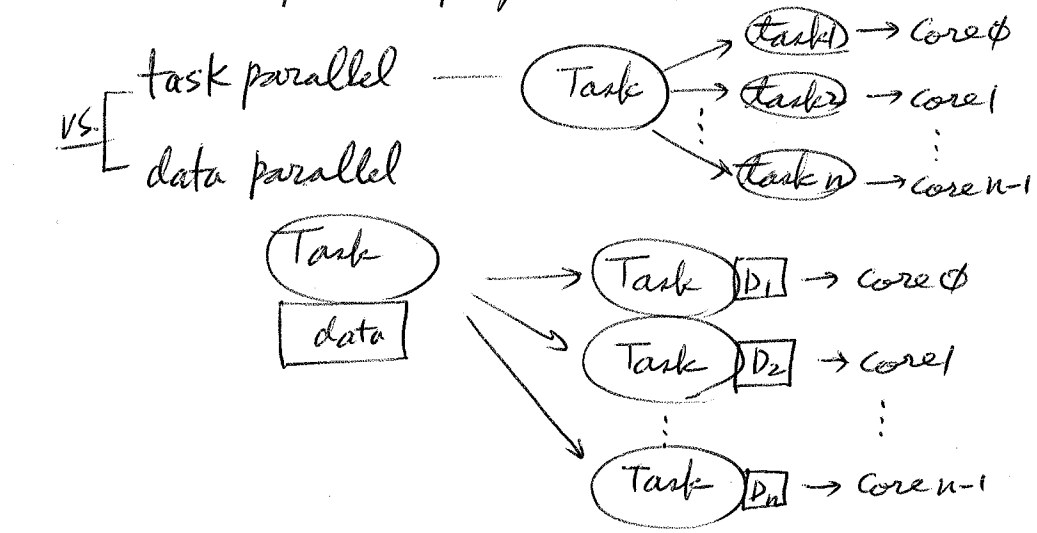
better way ↓

↓ better way (full-tree reduction)



— The gain is (with 8 cores)
7/3 = 2.33.. fold
with 1000 cores,
999/10 = 99.9 fold
($\log_2 1000 \approx 10$)

— How to write parallel programs ?

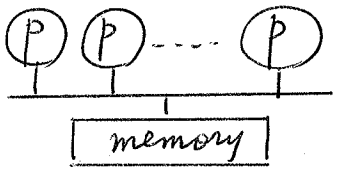


— issues — cores need coordination
- communication — send/receive msg, share info.
- synchronization — wait for proper order
- load balancing — reduce the critical path.

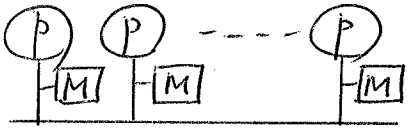
- parallel programming paradigms
  - vs.
    - using explicit parallel constructs (e.g., MPI, OpenMP)
    - higher-level parallel language — less efficient

- explicit way (more efficient)
  - C/C++ extensions with
    - libraries of type defs, funcs, macros
      - MPI — msg passing on distr.-mem system
      - pThread (POSIX Thread)
      - OpenMP — shared-mem. system (e.g., multicore)
        - library and some modifications to C/C++ compiler

- on shared-memory systems
  - pThread — lower level
  - OpenMP — higher level



- distributed-memory system



Terms (≠ clear boundary)

- Concurrent — multiple tasks progress simultaneously
- parallel — multiple tasks cooperate closely to solve a problem. time/HP are critical
- distributed — a program may need to cooperate with other programs to solve a problem
  - e.g.) client/server model