NOTASI UNTUK ALGORITMA PARALEL

- Untuk Shared-Memory Model
  Global
  Local

- Untuk Distributed Memory Machine
  Parameter → suatu konstanta yang sudah dikomunikasikan antar prosesor.

- Umum

  +, x, ←
  if … else … endif ; while … endwhile ; for … endfor
  for all … → data paralelism :
    - SIMD : diasumsikan ada lockstep
    - MIMD : diasumsikan berjalan asynchronous

- ⇐ menyatakan suatu prosesor store/retrieve nilai local dari prosesor lain.

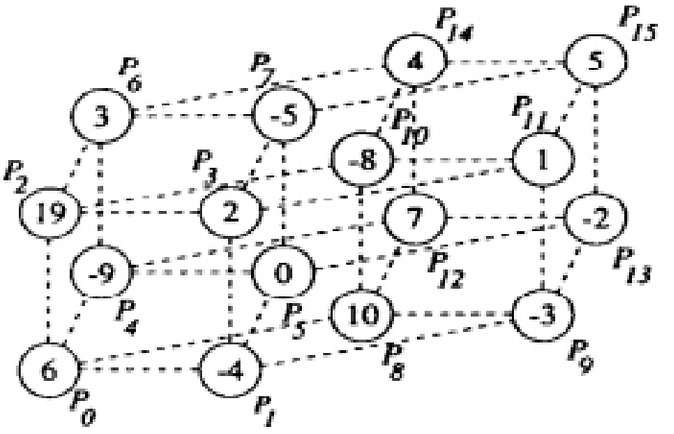
- [x] a menyatakan nilai dari prosesor.
  tmp ⇐ [s] a

SUMMATION (HYPERCUBE SIMD):

```
Parameter     n   { Number of  elements to add}
              p   { Number of processing elements}

Global        j
Local         local.set.size, local.value[1…⌈n/p⌉], sum, tmp

Begin
    For all P_i, where () ≤ i  ≤  p – 1 do
        If i < (n modulo p) then
            Local.set.size  ← ⌈n/p⌉
        Else
            Local.set.size  ← ⌊n/p⌋
        Endif
    Sum ← 0
     Endfor
     For j ← 1 to ⌈n/p⌉ do
        For all P_i, Where 0 ≤ i ≤  P-1 do
            If local.set.size ≥ j then
                Sum ← sum + local.value[j]
            Endif
        Endfor
    Endfor
    For j ← log P-1 downto 0 do
        For all P_i, Where 0 ≤ i ≤  P-1 do
            If i < 2^j  Then
                tmp ⇐ [ i+2^j] sum
                sum ← sum + tmp
            Endif
    Endfor
  Endfor
END
```



Values to be added

Hasilnya ?

Bagaimana jika setiap elemen pemrosesan akan mempunyai copy-an dari global sum ? Kita dapat menggunakan fase broadcast di akhir algoritma. Setiap elemen pemrosesan Θ mempunyai global sum, nilai-nilainya dapat dikirimkan ke processor lainnya dalam log p tahapan komunikasi dengan membalikan arah edge pada pohon binomial.

Kompleksitas untuk menemukan jumlah (sum) dari n nilai, **Θ (n/p + log p)** pada model hypercube yang berarray prosesor.

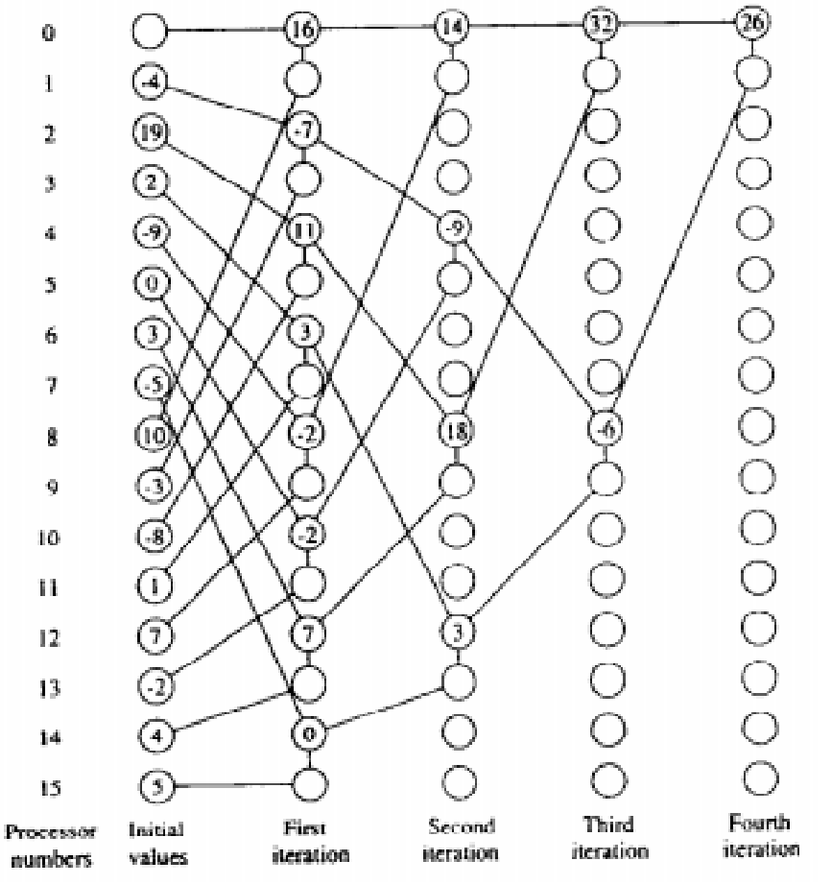
SHUFFLE – EXCHANGE SIMD MODEL

Pada model ini, tidal ada dilatasi – 1 pada pohon binomial. Efisiensi algoritma ini jika jumlah digabungkan secara berpasangan. Angka logaritma dari tahapan penggabungan dapat menemukan total akhir. Setelah log p shuffle exchange, prosesor O memdapatkan total akhir prosesor.

SUMMATION (SHUFFLE-EXCHANGE SIMD) :

```
Parameter     n { Number of elements to add }
              P { Number of processing elements }
Global        j
Local         local.set.size, local.value[1…⌈n/p⌉], sum,tmp
Begin
    For all Pi, where () ≤ i ≤  p – 1 do
        If i < (n modulo p) then
            Local.set.size  ← ⌈n/p⌉
        Else
            Local.set.size  ← ⌊n/p⌋
        Endif
     Sum ← 0
     Endfor
     For j ← 1 to ⌈n/p⌉ do
        For all Pi, Where 0 ≤ i ≤ P-1 do
            If local.set.size ≥ j then
                Sum ← sum + local.value[j]
            Endif
    Endfor
   Endfor
   For j ← 0 to log p-1 do
        For all Pi, Where 0 ≤ i ≤ P-1 do
            Shuffle (sum) ⇐ sum
            Exchange (tmp) ⇐ sum
            Sum ← sum + tmp
    Endfor
  Endfor
END
```

Waktu Kompleksitas : Θ (n/p + log p)



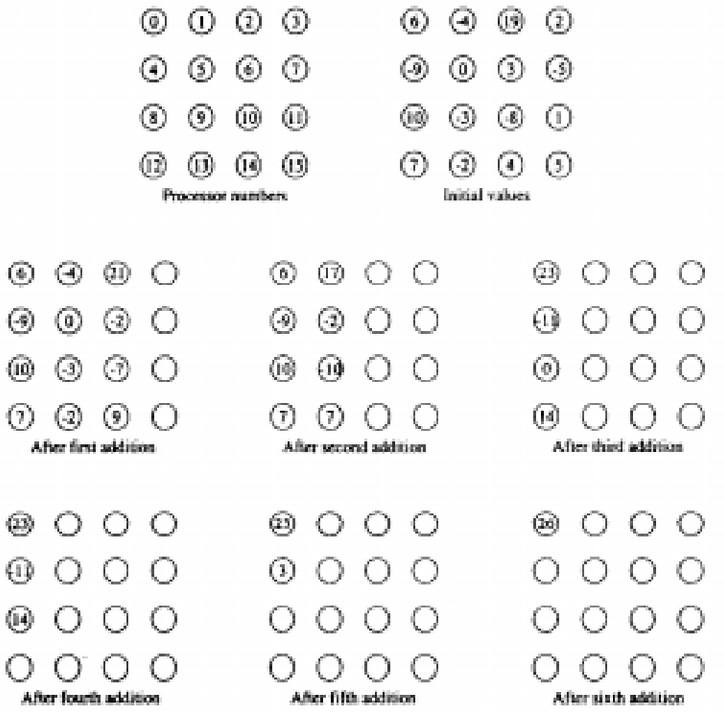
2-D Mesh SIMD Model

Untuk mendapatkan jumlah n nilai diantara p prosesor, diorganisasikan dalam **√p x √p** mesh, minimal satu prosesor pada mesh harus berisi total akhir.
Total jumlah dari tahap komunikasi untuk mendapatkan sub-total dari prosesor minimal **2(√p-1).**

Kompleksitas algoritma ini Θ(n/p + √p)

(Pseudocode (anggap n = l^2)
SUMMATION (2-D MESH SIMD) :

```
Parameter     l { Mesh has size l x l }
Global        i
Local         tmp,sum
Begin
   {each processor finds sum of its local values – code not
shown}
      For i ← l-1 downto 1 do
            For all Pj,i , Where 1 ≤ j ≤ l do
            {Processing elements in column i active}
                  tmp  ⇐ south (sum)
                  sum ← sum + tmp
            Endfor
      Endfor
END
```



Waktu Eksekusi Algoritma ini terbagi 2 :

- Waktu yang dibutuhkan untuk nilai awal pesan (*message – passing overhead*)
- Waktu yang dibutuhkan untuk melaksanakan pemindahan data (*message latency*)

Jika data yang di- *broadcast* sedikit, *message – passing overhead* mendominasi *message latency*.

Algoritma yang baik adalah yang meminimalkan jumlah komunikasi yang dilaksanakan oleh setiap prosesor. Jadi minimal untuk **komunikasi** membutuhkan ***log p.***

Pohon binomial cocok untuk model *broadcast* ini, karena ada sebuah dilatasi-1 yang ditempelkan ke *hypercube* dari pohon binomial (Lihat Gb. 6-11 dan 6 – 12)

Jika data yang di-broadcast besar, waktu pemindahan data mendominasi *message-passing overhead* . Pada pohon binomial-nya merupakan keadaan terburuk, dalam setiap waktu, tidak lebih dari p/2 dari p log p hubungan komunikasi yang digunakan.

Jika M adalah waktu yang dibutuhkan untuk melewatkan pesan dari satu prosesor ke lainnya, algoritma *broadcast* ini membutuhkan waktu M log p.

***Jhonson and Ho (1989*)** membuat algoritma *broadcast* yang mengeksekusi waktu lebih cepat dari algoritma bimial log p.

Algoritmanya : setiap hypercube terdiri dari "***log p edge-disjoint spanning tree"*** dengan node akar yang sama .
Algoritma memisahkan pesan ke dalam log p bagian dan broadcast setiap bagian ke node lain melalui pohon rentang binomial. (lihat Gb. 6-13)

Sehingga algoritma yang baru membutuhkan **waktu M log p/log p = M.**

HYPERCUBE BROADCAST (id,source,value) :

Parameter	P	{Number of processor}
Value	id	{Processor's id number}
	source	{ID of source processor}
Reference	value	{value to be broadcast}
Local	i	{loop iteration}
	partner	{partner processor}
	position	{position in broadcast tree}

{*This procedure is called from inside a for all statement*}

```
Begin
    position ← id ⊗ source
    For i ← 0 to log p-1 do
        If position < 2^i then
            partner ← id ⊗ 2^i
            [partner] value ← value
        Endif
    Endfor
END.
```

...together, to avoid repeated message startup

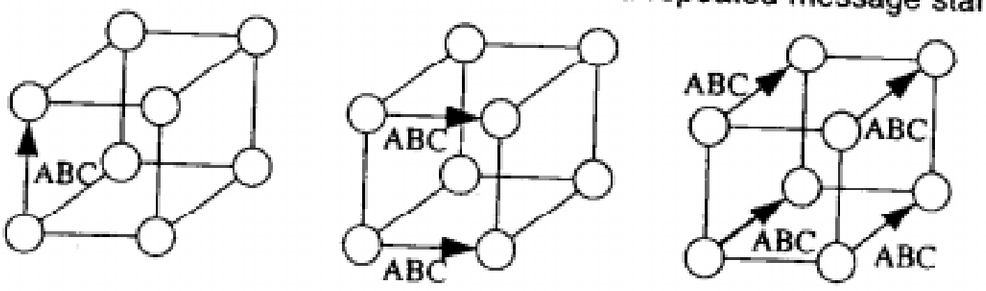


FIGURE 6-13 Johnsson and Ho's (1989) hypercube broadcast algorithm makes better use of the available communication links to improve performance when the message is long.