КС с общей памятью

Общая память
ШИНА
П1 П2 П3

+большая Общая память
+Организация связи узлов базируется на Общ. Памяти
+полно связная система – передача данных процессору за раз
-ШИНА – ограничение по пропускной способности
- наличие ОР

ПКС с локальной памятью

Проц+ЛП-----Проц+ЛП
| / |
Проц+ЛП--- Проц+ЛП

+нет ОР
-ЛП меньше ОП
-наличие дорогих линий связи
-больше времени на передачу данных

Распределенная КС

Комп Комп Комп Комп
| | | |

СЕТЬ

Коэффициент ускорения

Ку=Т1/Тп

Коэффициент загрузки

Кз=Ку/р

Концепция неограниченного параллелизма

- Процессов – добуя
- Т действий равны 1
- Т передачи данных = 0

Лемма Брента

Если Т время решения задачи из ЕН операций в параллельной системе с неограниченным числом процессоров, а Тр время решение в ПВС с конечным числом процессоров, то Тр = Т + (ЕН - Т)/р.

Теорема Мунро – Петерсона

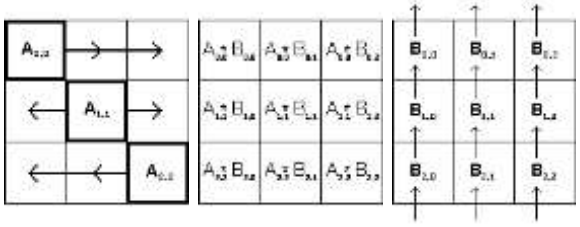
Если в ПВС с р – процессорами выполняется вычисление требующие ЕМ бинарных операций, то время вычисления ы этой системе:

$$tp = \frac{[\log_2 p]+[(m+1-2^{[\log 2p]})/p] \text{ для } m >= 2^{[\log 2p]}}{\text{Log}_2(m+1)}$$

Блочный алгоритм умножения матриц

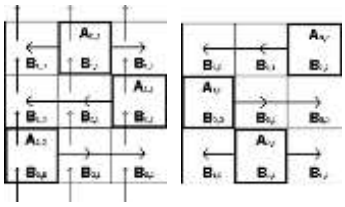
Рассмотрим работу Fox's алгоритма на примере умножения матриц 6-го порядка на 9-ти процессорах, то есть n=6, а p=9. В этом случае каждому процессору назначается подматрица порядка $n/(p^{1/2}) = 2$ от каждой из матриц A, B и C и Fox's алгоритм выполняет умножение матриц за $p^{1/2} = 3$ этапа:

Этап 0 (шаг 1 (слева), шаг 2 (по центру), шаг 3 (справа)):



На начальном этапе происходит рассылка подматриц $A_{i,i}$, стоящих на главной диагонали, процессорам, работающим с подматрицами в той же строке. Далее на каждом процессоре происходит умножение полученной диагональной подматрицы $A_{i,i}$ на подматрицу $B_{i,j}$, хранящуюся на данном процессоре. Результат умножения помещается в подматрицу $C_{i,j}$ процессора (i,j). Здесь i,j изменяются от 0 до 2. Перед переходом к следующему этапу происходит перемещение подматрицы $B_{i,j}$ от процессора (i,j) к процессору (i-1,j), то есть к непосредственно "верхнему" процессору. Процессоры нулевой строки посылают подматрицы $B_{0,j}$ процессорам последней (в данном случае второй) строки.

Этап 1 (слева) и этап 2 (справа):



На первом этапе также происходит рассылка, но только уже подматриц $A_{i,(i+1) \bmod q}$, где $q = p^{1/2} = 3$, а i изменяется от 0 до 2. То есть процессоры нулевой, первой и второй строк получат подматрицы $A_{0,1}$, $A_{1,2}$ и $A_{2,0}$ соответственно. Далее на каждом процессоре происходит умножение полученной подматрицы $A_{i,(i+1) \bmod 3}$ на подматрицу $B_{i,j}$, полученную на предыдущем этапе от процессора непосредственно нижней строки. Результат умножения складывается с подматрицей $C_{i,j}$ и снова в нее записывается. Перед переходом к следующему этапу снова происходит восходящее перемещение подматриц $B_{i,j}$, аналогичное их перемещению на этапе 0.

Второй (и в данном случае последний) этап работы Fox's алгоритма полностью аналогичен предыдущим этапам и может быть описан следующей последовательностью шагов:

- рассылка подматрицы $A_{i,(i+2) \bmod 3}$ процессорам i-той строки (на рисунке эти подматрицы выделены)
- умножение на процессоре (i,j) подматриц $A_{i,(i+2) \bmod 3}$ и $B_{i,j}$ (Понятно, что в общем случае, подматрицы $B_{i,j}$ на данном этапе и предыдущем не совпадают)
- $C_{i,j} = C_{i,j} + A_{i,(i+2) \bmod 3} * B_i$

	Семафоры	Мютексы	События	Критические секции	Мониторы
АДА	P(S) – Suspend_Until_True(S:in out SuspendObject) V(S) – Set_True(S)				Protected modul Private OP Entry «название входа» when «название барьера» = «значение открытия барьера» Procedure Signal изменяет барьер

Win32	S:HANDLE;
	S=CreateSemaphore(*,max,нач.значение,*)
	P(S)–WaitForSingleObject(S,Time(infinite))
	V(S)–ReleaseSemaphore(S,K);//S:=S+K
	WaitForMultipleObjects(MassivSemaforov,Time)
		Win32
		M = CreateSemaphore(*,нач.значение,*,*)
		P(M) –WaitForSingleObject(S,Time(infinite))
		V(M)–ReleaseMutex(S,K);
			E = CreateEvent(*,нач.значение,признак необх сбрасыв сигнала после прерывания,*)
			P(E) –WaitForSingleObject(E)
			V(E) –SetEvent(E);
				Z:Critical_Section
				Enter_Critrical_Section(z);
				…
				Exit_Critrical_Section(z);	

C#	Semaphore S=new Semphore(1,1) P(S) – S.waiteOne(); V(S) – S.releaseSemaphore();	Mutex M=new Mutex (false) P(M) – M.waiteOne(); V(M) – M.releaseMutex ();			Object x Monitor.Enter(x); Monitor.Exit(x);

Java				Synchronized(z){
				…
				}
					Private OP
					Synchronized
					методы доступа к ОР
					Wait() Notify()