MULTITHREADING – continued – Matrix Multiplication

```
#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define M 3  #define K 2 #define N 3

#define NUM_THREADS 10

 int A [M][K] = { {2,2}, {2,2}, {3,6} };

int B [K][N] = { {8,7,6}, {5,4,3} };

int C [M][N];

struct v { int i; /* row */ int j; /* column */ };

void *runner(void *param); /* the thread */
```

```
int main(int argc, char *argv[])
 { int i,j, count = 0;
for(i = 0; i < M; i++)
{ for(j = 0; j < N; j++)
{ //Assign a row and column for each thread
struct v *data = (struct v *) malloc(sizeof(struct v));
data->i = i; data->j = j;
/* Now create the thread passing it data as a parameter */ pthread_t
    tid; //Thread ID
pthread_attr_t attr; //Set of thread attributes
//Get the default attributes
pthread_attr_init(&attr);
//Create the thread
pthread_create(&tid,&attr,runner,data);
//Make sure the parent waits for all thread to complete
pthread_join(tid, NULL);
} }
```

```
//Print out the resulting matrix
for(i = 0; i < M; i++) { for(j = 0; j < N; j++)
{ printf("%d ", C[i][j]); } printf("\n"); }
}
//The thread will begin control in this function
void *runner(void *param)
{ struct v *data = param; // the structure that holds our data
int n, sum = 0; //the counter and sum
 //Row multiplied by column
for(n = 0; n< K; n++)
{ sum += A[data->i][n] * B[n][data->j]; }
//assign the sum to its coordinate
C[data->i][data->j] = sum;
//Exit the thread
pthread_exit(0); }
```

Amdahl's Law – Math of Multi Threading

- Gene Amdahl - a computer architect from IBM and Amdahl corporation)
- formula which gives the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected of a system whose resources are improved.
- **maximum improvement** possible by just improving a particular part of a system
- Speedup -- ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement
-

- --- ratio of execution time for the entire task without using the enhancement and execution time for the entire task using the enhancement.
- **If Pe** is the performance for entire task using the enhancement and **Pw** is the performance for entire task without using the enhancement then
  - **Speedup = Pe/Pw or Ew / Ee**
- **Fraction enhanced** –fraction of computation time in the original computer that is achieved thru enhancement (MT)
- if 10 seconds of the execution time of a program that takes 40 seconds in total can use an enhancement , **the fraction is 10/40**. -- *Fraction Enhanced.* Always less than 1

Amdahl's Law – Math of Multi Threading

- **Speedup enhanced** –improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program
- If the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. --- Speedup enhanced.
- *Speedup Enhanced is always greater than 1.*
- Moore's Law v/s Amdhal's Law v/s Niklaus Writh's law