Parallel Computer Architecture: Übung #1

Abgabe am Dienstag, 28. Oktober 2014

Günther Schindler, Fabian Finkeldey, Shamna Shyju

Inhaltsverzeichnis

Matrix-Vektor-Multiplikation

Im Zuge der ersten Übung ist ein Programm zur Matrix-Vektor-Multiplikation zu entwerfen. Nachfolgend werden Lösungen zu den einzelnen Anforderung dargestellt.
Die gezeigten Funktionen basieren auf Multiplikationen mit integer Werten. Da die Multiplikation ebenso mit double und float Werten vollzogen werden sollen, wurden hierfür ebenso Funktionen Implementiert.

Zunächst wird der, für die Matrizen verwendete, Datentyp vorgestellt. Via Typdefinition wird eine Struktur namens sMatrixInt erstellt. Ein Zeiger auf einen Zeiger gibt den jeweiligen Anfangspunkt der Matrix. Zusätzlich enthält die Struktur jeweils einen Wert für die Anzahl der Spalten und Zeilen.

```
typedef struct sMatrixInt
{
  int iRow;       // for rows
  int iCol;       // for columns
  int** ppaMat;   // int[][]
} sMatrixInt;
```

Die Größe der Matrix (und damit auch des Vektors) soll über Kommandozeilenparameter übergeben werden. Dadurch müssen die Matrizen dynamisch variabel sein. Die Funktion vAllocMatrixInt() alloziiert den für die Matrix benötigten Speicherplatz anhand des übergebenen Zeilen- und Spaltenwertes.

```
int vAllocMatrixInt(sMatrixInt *pM, int iRow, int iCol)
{
  int i=0, iRet=0;
  /* Init rows and cols in matrix structure */
  pM->iCol=iCol;
  pM->iRow=iRow;
  /* Alloc Mem for Rows-Pointer */
  pM->ppaMat = malloc(iRow * sizeof(int *));
  if(NULL == pM->ppaMat)
    iRet=1;
  /* Allocate Memory for Rows */
  for(i=0; i < iRow; i++)
  {
    pM->ppaMat[i] = malloc(iCol * sizeof(int *));
    if(NULL == pM->ppaMat[i])
      iRet=1;
  }
  return iRet;
}
```

Damit der alloziierte Speicherplatz wieder freigegeben wird, wird die Funktion vFreeMatrixInt() verwendet.

```
void vFreeMatrixInt(sMatrixInt *pM)
{
  int i;
  /* free the Rows */
  for(i=0; i<pM->iRow; i++)
    free(pM->ppaMat[i]);
  /* free cols */
  free(pM->ppaMat);
}
```

Die zu multiplizierenden Matrizen sollen durch die rand() Funktion initialisiert werden. Die Funktion vInitMatrixInt() übernimmt diese Initialisierung anhand des übergeben seed-Wertes, welcher bei Programmstart als Kommandozeilenparameter übergeben wird.

```
void vInitMatrixInt(sMatrixInt *pM, int iSeed)
{
  int i,j;
  /* Initializes random number generator */
  srand((unsigned) iSeed);
  /* Fill the matrix-elements with random numbers */
  /* matrix[zeile][spalte] */
  for(i=0; i<pM->iRow; i++)
  {
    for(j=0; j<pM->iCol; j++)
      /* Generate numbers fronm 0 to 50 */
      pM->ppaMat[i][j]=rand();
  }
}
```

Die eigentliche Matrix-Vektor-Multiplikation übernimmt die Funktion vMatrixVecMulInt(), der Matrix und Vektor als Parameter übergeben werden.

```
void vMatrixVecMulInt(sMatrixInt *pM, sMatrixInt *pV, sMatrixInt *pVRes)
{
  int i,j;

  /* Multiplying matrix by vector (vector=matrix[x][0]) */
  for(i=0; i<pM->iRow; i++)
  {
    pVRes->ppaMat[i][0]=0;
    for(j=0; j<pM->iCol; j++)
      pVRes->ppaMat[i][0] += pM->ppaMat[i][j] * pV->ppaMat[j][0];
  }
}
```

Schließlich sollen die Ausführungzeiten der Matrix-Vektor-Multiplikation gemessen werden.
Diese Messung soll zum einem anhand der gettimeofday() Funktion realisert werden. Nachfolgende Implementierung der Funktionen dstartMesGTOD() (starten der Messung) und dstopMesGTOD() (stoppen der Messung), wurden basieren auf der gettimeofday() Funktion realisiert. Die gemessene Zeit wird als return-Wert zurückgegeben.

```
double dstartMesGTOD(void)
{
  struct timeval tim;
  gettimeofday(&tim, NULL);
  return tim.tv_sec+(tim.tv_usec/1000000.0);
}

double dstopMesGTOD(double dStartTime)
{
  struct timeval tim;
  gettimeofday(&tim, NULL);
  return (tim.tv_sec+(tim.tv_usec/1000000.0)) - dStartTime;
}
```

Neben obiger Implementierung soll die Messung auch über den Assembler Befehl Read Time Stamp Counter (rdtsc) vollzogen werden. Wieder dient, wie folgend gezeigt, eine Funktion (ullstartMesRDTSC()) zum starten der Messung und eine Funktion (dstopMesRDTSC) zum stoppen der Messung. Da rdtsc() die Prozessorzyklen liefert, muss das Ergebniss noch durch die Prozessorfrequenz (hier 2.2 GHz) geteilt werden.

```
unsigned long long ullstartMesRDTSC(void)
{
  return rdtsc();
}

double dstopMesRDTSC(unsigned long long ullStartTime)
{
  unsigned long long ullStopTime;
  ullStopTime = rdtsc();
  /* The denominator depends on the CPU freq. In my case it is 2.2 GHz */
  /* To get the right value, run: lscpu on commandline */
  return (double)(ullStopTime - ullStartTime) / 2200000000;
}
```

Pi-Näherungsverfahren