Hochschule Harz **FB Automatisierung und Informatik**	Parallele Programmierung SoSe 2016
4. Labor: MPI	Thema: MPI-Programmierung im Cluster mit globalen Kommunikationsoperationen:

1 Versuchsziele

Einen Cluster aufbauen und Teilaufgaben an einzelne Rechner delegieren.

2 Grundlagen

MPI erlaubt die Parallelisierung von Algorithmen auf MIMD-Rechnern. Dazu werden die Daten vom "Master" einzelnen Tasks ("Slaves") zur Verfügung gestellt, lokale Berechnungen definiert und die Teilergebnisse an den Master zurückgeschickt.

MPI unterstützt diese Vorgehensweise durch Kommunikationsverfahren, die das Senden und Empfangen der Daten ermöglichen.

3 Versuchsaufgabe

Programmieren Sie die Simulation „Game of Life" von John Conway. Dazu wird eine beliebige Datei mit den Anfangsmustern vom Master eingelesen. Nach der ersten Ausgabe durch den Master, erhalten die Worker, inklusive des Masters, nun mittels der Funktion „Scatter" die einzelnen Bereiche. Sinnvoll wären hier die Zeilen. Nun tauschen die Worker untereinander die notwendigen Daten aus. Danach berechnen sie nach dem unten abgebildeten Algorithmus den neuen Zustand, siehe Seite 5. Mit Hilfe der Funktion „Gather" werden alle Teilergebnisse im Master zusammengefasst. Dieser gibt am Ende jedes Berechnungsschritts den aktuellen Stand aus.

4 Versuchsdurchführung

1) Erstellen Sie eine Win32-Konsolen-Programm
2) Kopieren Sie die Vorgabe „Aufgabe4.cpp". Fügen Sie die MPI-Optionen in das Projekt (siehe Anhang).
3) Herunterladen des Programms „mpi_start.exe" für ein 64-Bit-Betriebbsystem
4) Testen des Programms:
    - Übersetzen des Quellcodes
    - Starten des MPich-Programms (32 oder 64-Bit)

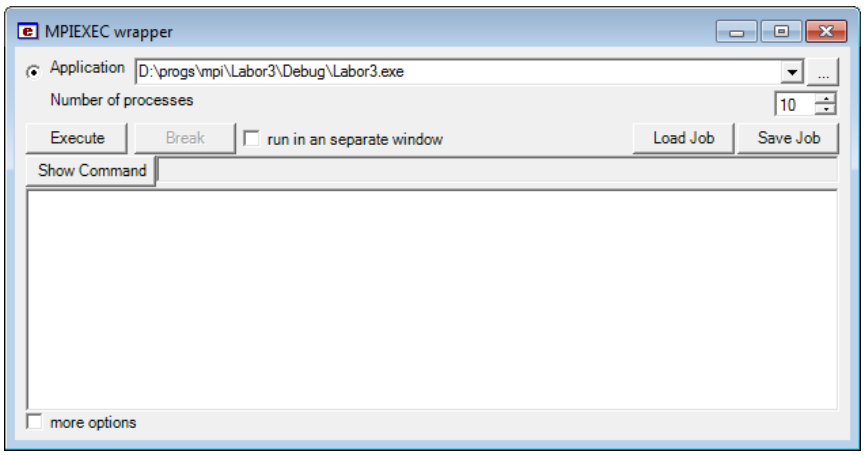


Abbildung 1 32-Bit

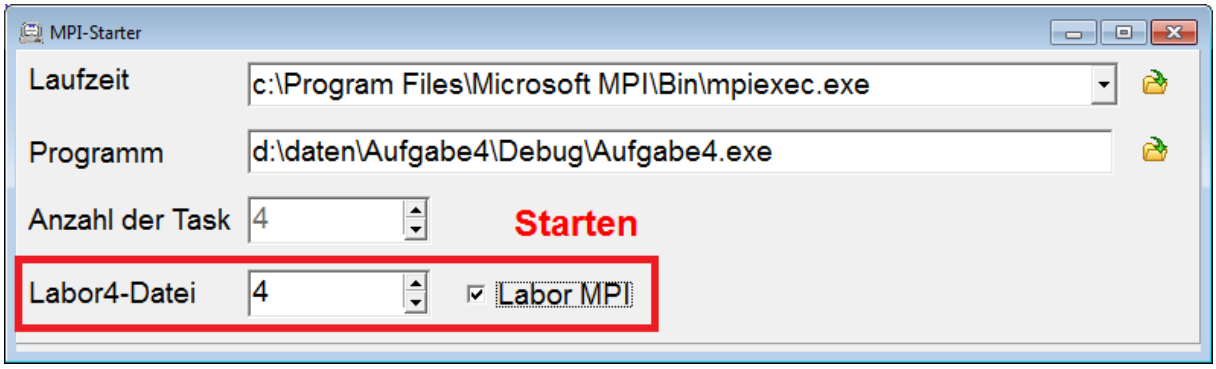


Abbildung 2 Starten des mitgelieferten Programms „mpi_start.exe“

Die Eingaben werden immer OHNE Apostrophe eingegeben.

- **Bitte die CheckBox „Labor MPI“ setzen.**
- **Damit wird die Game-of-Life-Variante ausgewählt UND automatisch die Anzahl der Worker.**

5) Kopieren Sie die Textdateien in das **erste** Debug-Verzeichnis (beinhaltet die Exe-Datei)
6) Folgende Module im Quellcode sind vorhanden:
    - readFile liest die aktuelle Datei ein, Aufruf nur vom Master
    - testPrintRoot Ausgabe des aktuellen Stands, Aufruf nur vom Master
    - testPrintLokal Ausgabe pro Worker
    - main Startroutinen
7) Die Wahl der Datei könnte man mittels eines „Konsolen-Menüs“ ermitteln, leider funktioniert dieses nicht mit der verwendeten MPI-Variante. Deshalb wird die Variable „wahl“ vorgegeben. Trotzdem kann man durch einen Parameter die Datei auswählen. Man kann im Aufruffenster einen zusätzlichen Parameter übergeben. Diese sollte 1 bis 8 sein. Dann kann man eine beliebige Dateie starten.
8) Das Einlesen der Datei und die Ausgabe der Werte wurden vorgegeben. Man muss es nur aufrufen.

9) Versenden Sie die Variablen „_anzCols“ und „_anzRows“ an alle Worker.

Der Master ist auch ein Worker

10) Nach dem Einlesen der Datei erhalten die Worker mit den globalen Variablen „_anzCols“ und „_anzRows“ die Abmessungen des Spielfeldes. Die Werte des Spielfeldes werden im Feld „_Root_zellen“ abgespeichert. Ein Stern symbolisiert ein Lebewesen. Mit den globalen Werten sollte man mehrere Plausibilitätsprüfungen durchführen.
11) Für das Berechnen bzw. Versenden benötigt man nun die Pid der angrenzenden Workers (prev, next). Für den ersten Test kann man die Variable „**_bDebug**“ auf true setzen. Damit werden die eingelesenen Zeilen mit speziellen Zeichen versehen.
    Wenn die Variable **_bDebug** gesetzt wurde, erscheint folgender Ausdruck:
    1) Ausgabe der Matrix:

        A0000
        B0000
        C1110
        D0100
        E1110
        F0000
        G0000
        H0000

    2) Ausgabe der empfangenen Zeilen

        Pid: 0 |A0000|
        Pid: 1 |B0000|
        Pid: 2 |C1110|
        Pid: 3 |D0100|
        Pid: 4 |E1110|
        Pid: 5 |F0000|
        Pid: 6 |G0000|
        Pid: 7 |H0000|

12) Mittels Scatter werden die Daten der Variablen „_Root_zellen“ verteilt.
13) Jeder Worker holt nun die Werte der benachbarten Worker, dabei ist der oberste Nachbar der letzte Eintrag. Und der untere Nachbar des letzten Workers ist der oberste Worker
14) Man berechnet nun den nächsten Schritt. Die Originalwerte dürfen hier noch nicht verändert werden. Dazu benötigt man ein zusätzliches Feld (lokal_Zellen_neu).
15) Mittels „Gather“ werden alle Teilergebnisse im Root zusammengefasst. Nun wird der neue Zustand vom Root ausgegeben.
16) Da der Lebenszyklus pro Datei unterschiedlich ist, muss am Ende ermittelt werden, ob es mindestens eine Änderung in einem der Worker gab. Dazu vergleicht man mit der Funktion **strcmp** den Originalstring mit dem neuen String. Gab es eine Änderung, wird die lokale variable „`l_change`“ auf eins gesetzt. Andernfalls erhält sie den Wert null. Mittels der „Reduce“-Anweisung werden alle „`l_change`“ gesammelt. Hat die Summe den Wert Null, kann der Master die Schleife mittels break verlassen. **ABER** auch die restlichen Worker müssen automatisch beendet werden. Dazu sendet der Master vor dem break die Summe an alle Worker („BCast“). Danach beenden alle Client zusammen die Schleife.

I

Anzahl der Zeilen in der Musterlösung: 385 Zeilen

Anzahl der Worker pro Datei:

- File1.txt 8 Worker
- File2.txt 8 Worker
- File3.txt 10 Worker
- File4.txt 10 Worker
- File5.txt 10 Worker
- File6.txt 36 Worker
- File7.txt 30 Worker
- File8.txt 34 Worker

5 Game of Life

Beschreibung:
Das Spiel des Lebens ist ein vom Mathematiker John Horton Conway 1970 entworfenes System, basierend auf einem zweidimensionalen zellulären Automaten. Jedes Gitterfeld ist ein Zellulärer Automat (Zelle), der einen von zwei Zuständen einnehmen kann(tot oder lebendig) bezeichnet werden. Als erstes wird eine Anfangsgeneration von lebenden Zellen auf dem Spielfeld platziert. Jede lebende oder tote Zelle hat auf diesem Spielfeld genau acht Nachbarzellen. Die nächste Generation ergibt sich durch die Befolgung einfacher Regeln.

Spielregeln:
Die Folgegeneration wird für alle Zellen gleichzeitig berechnet und ersetzt die aktuelle Generation. Der Zustand einer Zelle, lebendig oder tot, in der Folgegeneration hängt nur vom Zustand der acht Nachbarzellen dieser Zelle in der aktuellen Generation ab.

Die Regeln:
- Eine tote, leere, Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Eine lebende Zelle kann nur weiterleben, wenn sie zwei oder drei lebende Nachbarn hat. Bei einem Nachbarn stirb sie an Einsamkeit. Bei mehr als drei Nachbarn stirbt sie an Überbevölkerung
- Gezählt werden vertikale, horizontale und diagonale Richtungen
- Eine Zelle am linken Rand hat nicht den Nachbarn ganz rechts

Beispiel:

Ausgangsbild:

	*****	*****	*****

Folgebild:

		*****	
		*	
		*	

Aufbau der Datei:

```
10                  Anzahl der Spalten
8                   Anzahl der Zeilen
0000000000          Spielfeld
0000000000
0000000000
0000000000
0000111000
0000000000
0000000000
0000000000
```

6 Aufruf

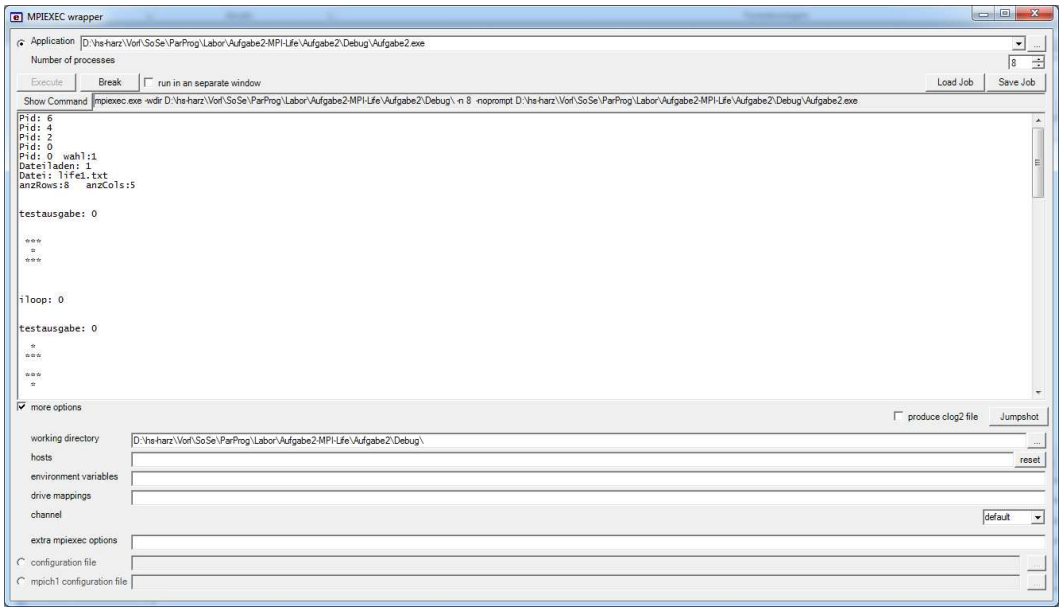
Aufruf:

C:\Daten\Labor\Aufgabe4-MPI-Life\Aufgabe4\Debug\Aufgabe4.exe 1

Startet die Datei „life1.txt“

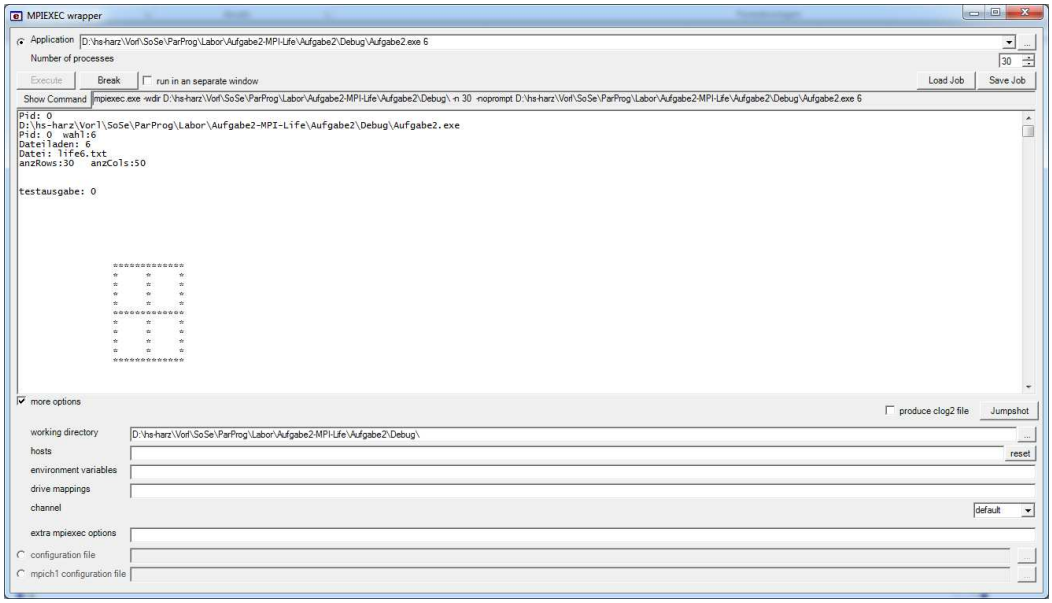
C:\Daten\Labor\Aufgabe4-MPI-Life\Aufgabe4\Debug\Aufgabe4.exe

Startet die vorgegebene Datei „life1.txt“



C:\Daten\Labor\Aufgabe4-MPI-Life\Aufgabe4\Debug\Aufgabe4.exe 6

Startet die Datei „life6.txt“



Beachten Sie die Dateien mit den Ergebnissen !!!!!!!

7 Kommunikationsfunktionen

Folgende Funktionen existieren:

- Comm_size
- Send
- Recv
- MPI_Bcast Einer an alle
- MPI_Reduce Sammeln von Werten der Worker zum Root
- MPI_Gather Alle an einen, Felder
- MPI_Scatter Einer an alle, Aufteilen von Feldern

8 Syntax der MPI-Anweisungen:

Bestimmen der Anzahl der Knoten

MPI_Comm_size:
Liefert die Größe des angegebenen Kommunikators; dh. die Anzahl der Prozesse in der Gruppe

Syntax: MPI_Comm_size(comm, &size)
Eingabe: comm - Kommunikator (handle)
Ausgabe: size - Anzahl der Prozesse in der Gruppe von comm (integer)

Anmerkungen:
MPI_COMM_NULL (der 'leere' Kommunikator) ist kein gültiges Argument für diese Funktion.

Bestimmen der aktuellen Pid

Funktion MPI_Comm_rank:

Bestimmt den Rang, Pid, des rufenden Prozesses innerhalb des Kommunikators

- Der Rang wird von MPI zum Identifizieren eines Prozesses verwendet.
- Die Rangnummer ist innerhalb eines Kommunikators eindeutig.
- Dabei wird stets von Null beginnend durchnumeriert.
- Sender und Empfänger bei Sendeoperationen oder die Wurzel bei kollektiven Operationen werden immer mittels Rang angegeben.

Syntax:: MPI_Comm_rank(comm, rank)
Eingabe: comm - Kommunikator (handle)
Ausgabe: rank - Rang des rufenden Prozesses innerhalb von comm (integer)

Senden:(blockierend, synchron)
Sender wartet, Empfänger muss noch nicht bereit sein

```
int MPI_Send(   void  *smessage,        // sendepuffer
                int count,              // Anzahl der Variablen
                MPI_Datatype datatype,
                int dest,               // Pid des Zielrechner
                int tag,                // Wunsch-Tag, meist 0
                MPI_Comm comm);         // Bereichsname
```

```
retcode = MPI_Send(….);
if ( retcode!= MPI_SUCCESS) {
        puts("Fehler beim Aufruf von Send");
}
```

Empfangen:(blockierend, synchron)
Empfänger wartet, Sender muss noch nicht gesendet haben

```
int MPI_Recv(   void  *rmessage,        // Referenz auf einen Empfangspuffer
                int count,              // Anzahl der Variablen
                MPI_Datatype datatype,
                int source,             // Pid des Sendeprozesses
                int tag,                // Wunsch-Tag, meist 0
                MPI_Comm comm,
                MPI_Status *status);    // Struktur der Rückgabe
```

Rückgabewerte:

```
retcode = MPI_Recv(….);
if ( retcode!= MPI_SUCCESS) {
        puts("Fehler beim Aufruf von Recv");
}
```

Werte der Struktur:

```
status.MPI_SOURCE
status.MPI_TAG
status.MPI_ERROR
```

8.1.1 Datentypen

MPI Datentyp	C Datentyp
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_INT ???	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_BYTE	

Globales Senden:

```
int MPI_Bcast(  void  *smessage,          // Referenz auf den Sendepuffer
                int count,                // Anzahl der Werte
                MPI_Datatype datatype,
                int root,                 // Sender, meist Root
                MPI_Comm comm);           // Bereichsname
```

Sender „root“ liefert die Nachricht
Alle anderen sind Empfänger

Rückgabewerte:

```
retcode = MPI_Recv(….);
if ( retcode!= MPI_SUCCESS) {
      puts("Fehler beim Aufruf von Recv");
}
```

Sammeln von lokalen Werten zum Root:

```
int MPI_Reduce(void* operand          // lokale Variable zum Summieren, Max, Min
               void* result,          // Root-Variable zum Summieren, Max, Min
               int count,             // Anzahl der Werte
               MPI_Datatype datatype, // Datentyp
               MPI_Op operator,       // Operator
               int root,              // Wer sammelt?, meist Root
               MPI_Comm comm);        // Bereichsname
```

Sender „root“ liefert die Nachricht
Alle anderen sind Empfänger

Beispiel und Rückgabewerte:

```
retcode = MPI_Reduce(&local, &root, 1, MPI_INT, MPI_MAX, 0,  MPI_COMM_WORLD);
```

```
if ( retcode!= MPI_SUCCESS) {
        puts("Fehler beim Aufruf von Reduce");
}
```

MPI Operation	**Mathematische Operation**
MPI_MAX	**Maximum**
MPI_MIN	Minimum
MPI_SUM	**Summe**
MPI_PROD	Produkt
MPI_LAND	Logisches UND
MPI_BAND	Bitweise UND
MPI_LOR	Logisches ODER
MPI_BOR	Bitweise ODER
MPI_LXOR	Logisches exklusives Oder
MPI_BXOR	Bitweises exklusives Oder
MPI_MAXLOC	Maximaler Wert und deren Index
MPI_MINLOC	Minimaler Wert und deren Index

Teilen eines Feldes:

```
int MPI_Scatter(
        void* sendbuf,              // Referenz auf den Sendepuffer des Roots
        int sendcount,              // Anzahl der lokalen Werte
        MPI_Datatype sendtype,      // Datentyp
        void* recvbuf,              // Referenz auf den Empfangspuffer in den Workern
        int recvcount,              // Anzahl der lokalen Werte
        MPI_Datatype recvtype,      // Datentyp
        int root,                   // Sender
        MPI_Comm comm)
```

Zusammenfassen eines Feldes:

```
int MPI_Gather(
        void* sendbuf,              // Referenz auf den Sendepuffer der Worker
        int sendcount,              // Anzahl der lokalen Werte
        MPI_Datatype sendtype,
        void* recvbuf,              // Referenz auf den Sendepuffer des Roots
        int recvcount,              // Anzahl der lokalen Werte
        MPI_Datatype recvtype,      // Datentyp
        int root,
        MPI_Comm comm)
```

Rückgabewerte:

```
retcode = MPI_Recv(….);
if ( retcode!= MPI_SUCCESS) {
        puts("Fehler beim Aufruf von Gather");
}
```

Zufallszahlen:

a) Initialisierung

srand (time(NULL)+Pid);

b) Eine Zufallszahl in Bereich 0 bis RAND_MAX erzeugen
int x= rand()

9 Literatur

- Vorlesungscripte

- **Rauber, Rünger:**
Parallele und verteilte Programmierung, Springer Verlag, 2000, ISBN 3-540-66009-7

- **Theo Ungerer,**
Parallelrechner und parallele Programmierung, Spektrum Akademischer Verlag, 1997, 3-8274-0231-X

- **Ananth Gama, Anshul Gupta:**
Introduktion to Parallel Computing; ISBN: 0-201-64865-2

- Beispiele auf meiner Internetseite

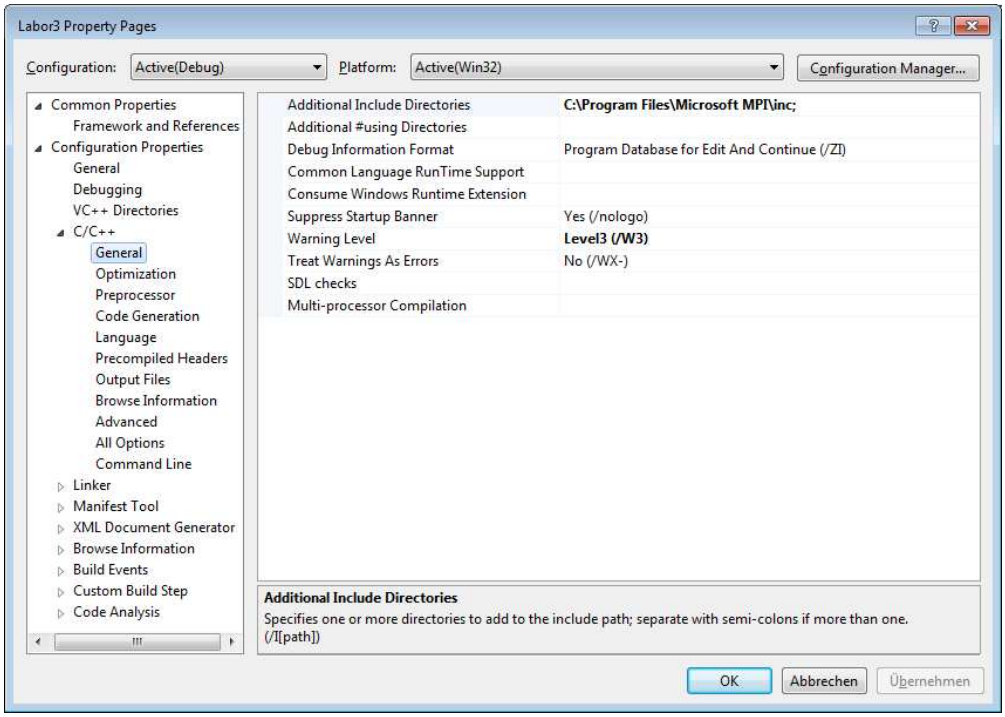
- http://www.via-claudia-rs.de/page/mitte/methoden/monte%20carlo/

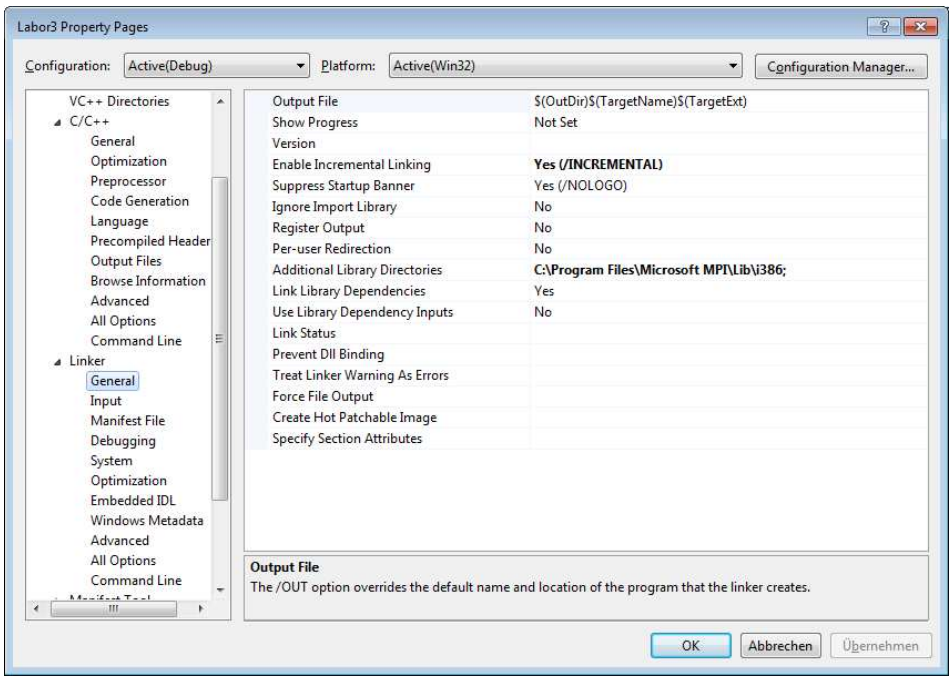
- http://random.mat.sbg.ac.at/~peter/students/rng.html#start

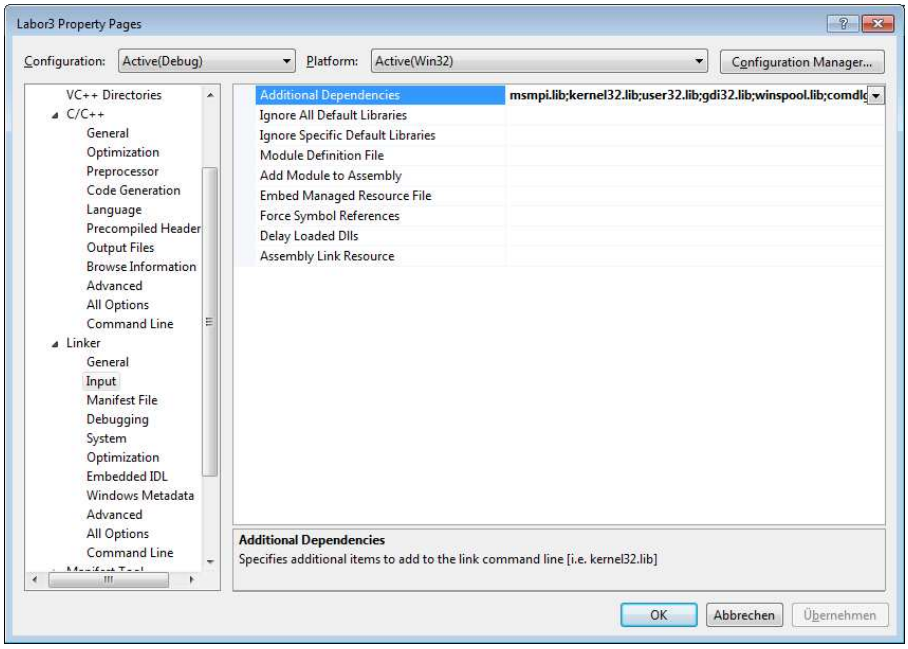
10 Installation MPI unter Visual Studio 2012

- Projekt anlegen (Win32, Console).
- **Löschen der Checkbox “Precompiled Header“**
- **Löschen der Checkbox “Security Development Lifecycle**“
- Kopieren des Rahmenquellcodes „Aufgabe3.cpp“
- Taste F6/F7 bzw. Strg+Alt+B erzeugt Projekt
- Taste **Alt+F7** erzeugt Eigenschaftsfenster
- Eintragen des include-Verzeichnisses
  - 32 Bit D:\Daten\MPICH2\include\
  - 64 Bit: C:\Program Files\Microsoft MPI\inc;
- Eintragen des lib-Verzeichnisses
  - 32 Bit D:\Daten\MPICH2\lib\
  - 64 Bit C:\Program Files\Microsoft MPI\Lib\i386;
- Eintragen der MPI-Libray unter Linker-Register:
  - 32 Bit: mpi.lib
  - 64 Bit: msmpi.lib;
- Übersetzen mit den Tasten Strg+Alt+B
- Aufruf der MPI-Konsole
- Starten (Schalter Run)

64: Bit:




Labor3 Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Header
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Output File $(OutDir)$(TargetName)$(TargetExt)
Show Progress Not Set
Version
Enable Incremental Linking Yes (/INCREMENTAL)
Suppress Startup Banner Yes (/NOLOGO)
Ignore Import Library No
Register Output No
Per-user Redirection No
Additional Library Directories C:\Program Files\Microsoft MPI\Lib\i386;
Link Library Dependencies Yes
Use Library Dependency Inputs No
Link Status
Prevent Dll Binding
Treat Linker Warning As Errors
Force File Output
Create Hot Patchable Image
Specify Section Attributes
Output File
The /OUT option overrides the default name and location of the program that the linker creates.
OK
Abbrechen
Übernehmen


Labor3 Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Header
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Additional Dependencies msmpi.lib;kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg
Ignore All Default Libraries
Ignore Specific Default Libraries
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded Dlls
Assembly Link Resource
Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]
OK
Abbrechen
Übernehmen

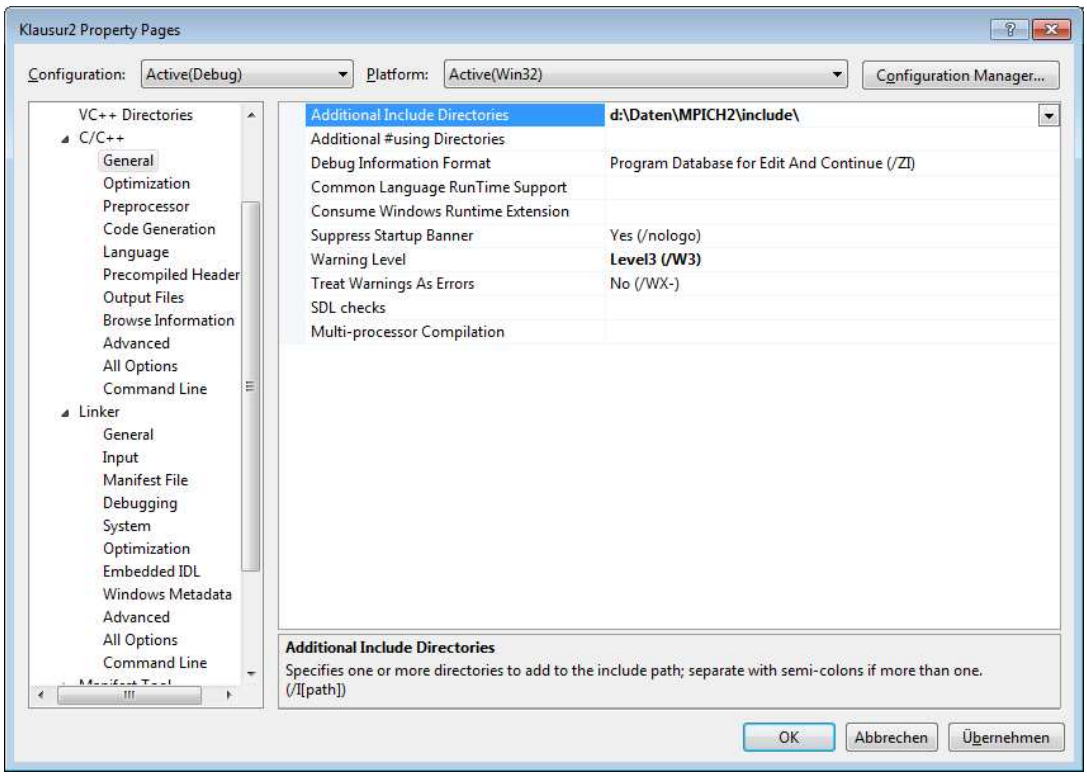


Abbildung 3 Setzen der Include-Datei

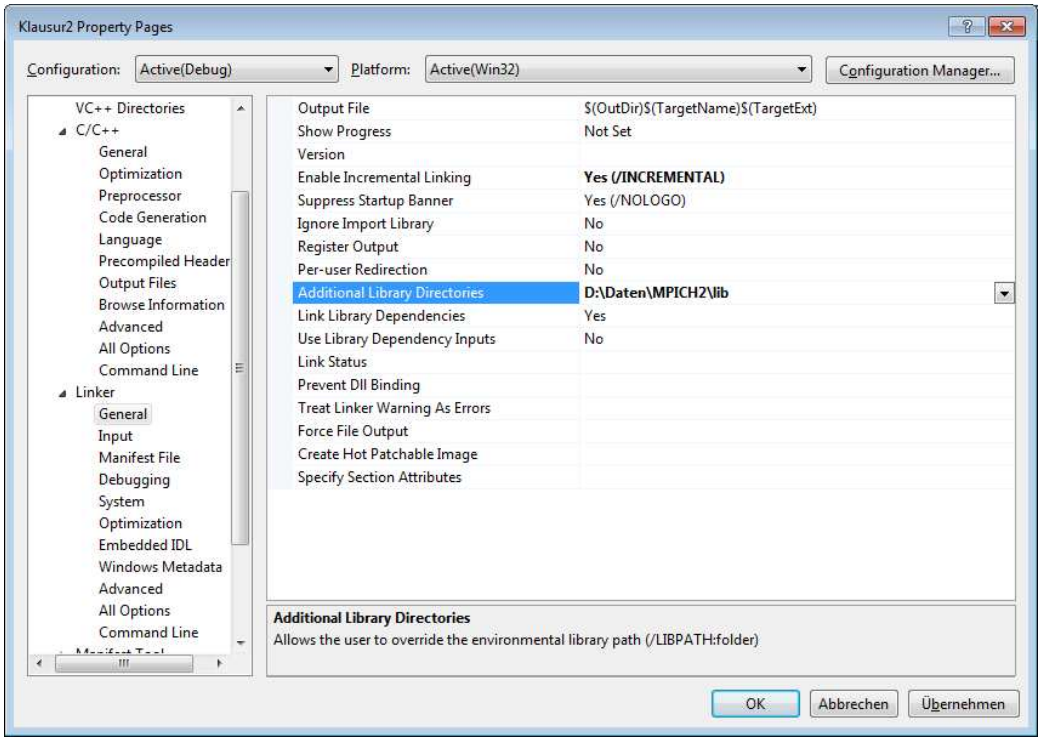


Abbildung 4 Setzen des Library-Verzeichnisses

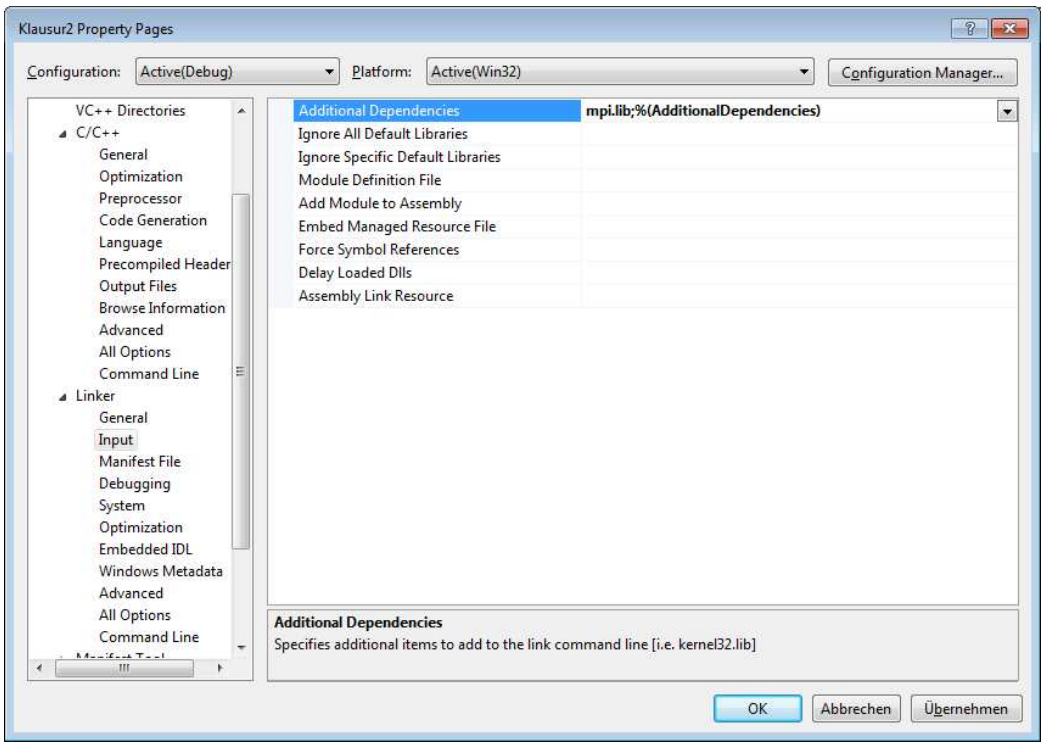


Abbildung 5 Eintragen der MPI-Library MPI.lib

Des Weiteren muss man den inkrementellen Linker ausschalten.

Aufruf des MPI-Passwort-Programms

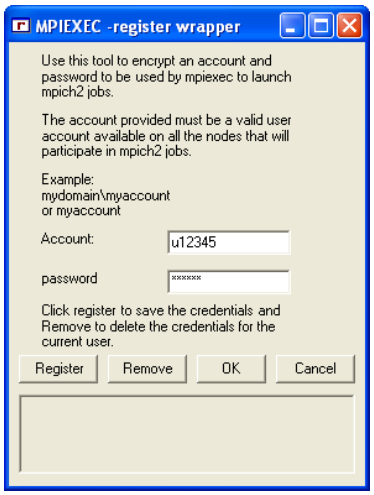


Abbildung 6 Aufruf von „C:\Programme\MPICH2\bin\wmpiregister.exe“