2.1 Teilprojekt B2

Objektorientierte parallele FEM-Systeme

2.1.1 Antragsteller

Prof. Dr.–Ing. W. Rehm

Fakultät für Informatik
Technische Universität Chemnitz
09117 Chemnitz

Tel.: (0371) 531 1420
email: rehm@informatik.tu-chemnitz.de

2.1.2 Projektbearbeiter

Dipl.-Math. L. Grabowsky

Fakultät für Informatik
Technische Universität Chemnitz
09117 Chemnitz

(von 01/1996 bis 10/1997)

Dipl.-Inf. (FH) M. Meyer

Fakultät für Informatik
Technische Universität Chemnitz
09117 Chemnitz

Tel.: (0371) 531 1408
email: marme@informatik.tu-chemnitz.de

(seit 10/1997)

2.2 Ausgangsfragestellung/Einleitung

Portable Simulationssoftware ist komplex. An Entwicklungen nehmen meist verschiedene Personen über einen längeren Zeitraum teil.

Komplexität und Weiterentwickelbarkeit erfordern strukturierte Software und eine geeignete Softwaretechnologie.

Diesbezüglich stellt die *objektorientierte Softwareenwicklung* im sequentiellen Rechnen eine anerkannte Lösungsmethode dar.

Kann dieses Paradigma im Bereich paralleler, wissenschaftlich–technischer Software ebenfalls erfolgreich angewendet werden?

Objektorientierte Programmierung im allgemeinen zielt auf *Portabilität*, *Modularität*, *Erweiterbarkeit* und *Wartbarkeit* ab.

Wie sieht die Erreichbarkeit dieser Ziele im konkreten Fall des Programmoduls SPC–PM Po2D (2D–Potentialprobleme) aus? Insbesondere, wie kann trotz Portabilität eine akzeptable Laufzeiteffizienz gewährleistet werden?

Portabilität wird nicht nur durch die Verwendung eines geeigneten *Sprachstandards* – hier C++ – erreicht; praktisch eng verzahnt damit steht die Verwendung eines (quasi) Standards für entsprechende *Kommunikationsstrukturen*, um von der Implementierung auf einem konkreten Parallelrechner unabhängig sein zu können.

So ist es wichtig, in allen Phasen der OO–Umsetzung eines Programmsystems beide Aspekte zu berücksichtigen und verschiedene Möglichkeiten und Varianten gleichermaßen im Auge zu behalten, zu testen und zu bewerten.
Diesem Ansatz folgend wurden – neben der objektorientierten Implementierung des Programmsystems – insbesondere auch portable, Effizienz–steigernde bzw. – erhaltende Kommunikationsmechanismen untersucht.

2.3 Forschungsaufgaben/Methoden

2.3.1 Einordnung der Teilaufgaben

Zunächst wurden die vorliegenden FEM–Systeme in enger Zusammenarbeit mit den Entwicklern umfassend analysiert; dies beinhaltete die Klärung von Fragen zu den verwendeten (Kommunikations–) Algorithmen, den zugrunde liegenden Datenstrukturen und der internen Speicherverwaltung des vorliegenden FORTRAN–Quelltextes.
Neben der in 2.2 geforderte Portabilität der untersuchten FEM-Bibliotheken leiten sich aus den bei der Analyse erworbenen Kenntnissen vor allem zwei Teilaufgaben aus der Gesamtproblematik ab:

1. Portable, architekturunabhängige Kommunikation in Multiprozessorumgebungen; Entwicklung geeigneter und effizienter Algorithmen auf der Basis von hardwareunabhängigen Kommunikationsbibliotheken.

2. Portable Programmierung durch Verwendung des Sprachstandards C++, für den auf den entsprechenden Plattformen Compiler verfügbar sind, um die Forderungen nach Modularität, Erweiterbarkeit und Wartbarkeit zu erfüllen.

Die Lösung jeder Teilaufgabe trägt gleichermaßen zur Erfüllung der Forderung nach Laufzeiteffizienz bei, indem sowohl die Art und Weise der Kommunikation, als auch die verwendeten Datenstrukturen und Algorithmen in dieser Hinsicht optimiert werden.

2.3.2 Teilaufgabe Portable Kommunikationsmechanismen für (adaptive) FEM–Systeme

2.3.2.1 Kommunikation in den Programmoduln *SPC–PM Po2D* und *SPC–PM Po3D*

Die Kommunikation in den an der Fakultät für Mathematik der TU Chemnitz entwickelten FEM–Systemen *SPC–PM Po2D* (2D–Potentialprobleme) bzw. *SPC–PM Po3D* (3D–Potentialprobleme) setzt auf einer speziellen Kommunikationsbibliothek, der sogenannten Cubecom–Bibliothek [SPC-9520], auf, in der die Kommunikation auf der Hypercube–Topologie basiert.

Ziel dieser Bibliothek ist es, kollektive Operationen weitestgehend hardwareunabhängig zu implementieren und nur jeweils eine simple Sende– bzw. Empfangsroutine tatsächlich auf die gegebene Maschine abzubilden. Speziell bei der sogenannten Koppelrandkommunikation auftretende Kommunikationsstrukturen bereiteten jedoch erhebliche Probleme und bedurften in den Programmsystemen einer Sonderbehandlung. Sowohl diese algorithmischen Schwierigkeiten, als auch die unbefriedigende Effizienz der Kommunikationsroutinen machten die Untersuchung der Eignung von MPI (*Message Passing Interface* [MPI95]) zur Vereinfachung und Systematisierung der anfallenden Kommunikationsanforderungen dringend erforderlich.

2.3.2.2 MPI–Implementierung der Cubecom–Bibliothek

In einem ersten Schritt wurden einzig die primitiven Sende– und Empfangsroutinen durch entsprechende MPI–Funktionen ersetzt [BEY95].

Neben einer deutlichen Effizienzsteigerung aller Kommunikationen wurde mit der zunehmenden Verfügbarkeit von MPI für die wichtigsten Hardwareplattformen gleichzeitig eine Unabhängigkeit vom Maschinentyp erreicht; die Portierbarkeit der FEM–Systeme wurde unproblematisch.

In einer anschließenden Arbeit [ERM96] erfolgte die Umstellung der Cubecom–Funktionalität auf den MPI–Mechanismus. Drei grundsätzliche Kommunikationstypen lassen sich dabei unterscheiden:

- *Prozeß–zu–Prozeß–Kommunikation*

  Im Originalsystem fanden zwei Routinen zur Kommunikation in einer Ringstruktur Anwendung, die das Aus– bzw. Einlesen großer Datenmengen unterstützten. Der Ring diente dabei der Kontrolle der auf Prozessor 0 zu sammelnden (bzw. auszuteilenden) Datenmenge, um einen Überlauf des Speichers zu verhindern. MPI überwacht derartige Situationen selbstätig, so daß sich die Verwendung eines Ringes erübrigte und die Handhabung vereinfachte.

- *Globale Kommunikation*

  Broadcasts, Scatter– und Gatherfunktionalität werden durch entsprechende MPI–Primitive realisiert, insbesondere entfällt die in der Originalbibliothek notwendige Linkberechnung.

- *Reduktionsoperationen*

  MPI bietet die volle Funktionalität, um globale Summenbildung, Produktbildung, globales Minimum und Maximum effizient zu realisieren.

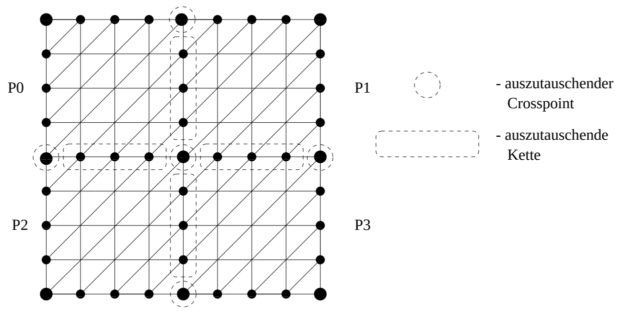
Durch MPI wurde das hypercube–basierte Modell des Originalsystems abgelöst, so daß die FEM–Simulation auch auf Prozessoranzahlen möglich wurde, die keinen Hypercube beschreiben.

2.3.2.3 Koppelrandkommunikation mittels des *Kommunikatorprinzips*

Bis auf die Routinen zur Koppelrandkommunikation (bzw. zur Berechung der Koppelflächen im 3D–Fall) setzten alle Routinen direkt auf die Cubecom–Bibliothek auf; in weiteren Arbeiten ([RATR-0297], [SFB-9717]) wurde nun eine neue Realisierung der Koppelrandkommunikation vorgestellt, auf die nachfolgend näher eingegangen werden soll.

Zum Verständins der nachfolgenden Ausführungen werden einige Bezeichnungen benötigt, die für das betrachtete Originalsystem [SPC-9520] gewählt wurden. In diesem entsteht die endgültige Vernetzung dadurch, daß ein gegebenes Grobnetz mehrfach verfeinert wird. Die Knoten des Grobnetzes werden als *Crosspoints* bezeichnet; die bei der Verfeinerung des Netzes zwischen den Crosspoints entstehenden Knoten werden zu Ketten zusammengefaßt. Abbildung 2.1 zeigt die Situation für ein auf 4 Prozessoren aufgeteiltes Gebiet mit den zwischen den Prozessoren auszutauschenden Crosspoints und Ketten.

Abbildung 2.1: Beispielgebiet für 4 Prozessoren.



Unter Verwendung von MPI leitet sich daraus folgende Realisierungsvariante ab: Für alle (möglicherweise) auszutauschenden Teilvektoren (also Ketten und Crosspoints) wird eine global eindeutige Identifikationsnummer festgelegt.

Aus allen Teilvektoren, die zwischen genau denselben Prozessoren auszutauschen sind, wird ein MPI–Datentyp erzeugt und aus den beteiligten Prozessoren eine Prozessorgruppe im Sinne von MPI gebildet, aus welcher anschließend ein Kommunikator (⇒ *Kommunikatorprinzip*) erzeugt werden kann. Innerhalb eines solchen Kommunikators können die Teilvektoren verteilt und gleichzeitig deren Inhalte aufsummiert werden. Um die Anzahl sequentieller Kommunikationschritte minimal zu halten, ist nur für eine geeignete Anordnung der Kommunikatoren in sogenannten Linkleveln zu sorgen.

Mittels des Kommunikatorprinzips wird die Koppelrandkommunikation vollständig formalisierbar und eine hardwareabhängige Implementierung vermieden. Der enormen Effizienzsteigerung bei der Kommunikation selbst (vor allem im 3D–Fall) steht allerdings ein Mehraufwand zur Erzeugung der globalen Identifikationsnummern aller Teilvektoren und zur Bildung der Kommunikatoren und Datentypen gegenüber, für die eigene (zeitaufwendige) Routinen notwendig wurden. In dem in Abschnitt 2.3.3.5 beschriebenem Netzgenerator werden alle notwendigen Informationen zu Kommunikationsbeziehungen allerdings bereits bei der Netzgenerierung gewonnen, so daß ersterer Nachteil in der objektorienterten Implementierung behoben sein wird. Die Routinen zur Kommunikatorenbildung wurden in [SFB-9810] überarbeitet und erreichen zumindest bei kleineren Verfeinerungsleveln (≤ 3) in Sachen Effizienz die Routinen, die im Orignalsystem zur Vorbereitung der Koppelrandkommunikation notwendig waren (siehe 2.5.1).

2.3.2.4 Einfluß von Partitionierungsstrategien auf die Kommunikation

In [SFB-9717] wurde untersucht, inwiefern sich unterschiedliche Partitionierungsstrategien auf das Kommunikationsverhalten bei 3D–FEM–Systemen auswirken. Gegenübergestellt wurden die jeweils schnellsten Implementierungen des Programmpakets *SPC–PM Po3D* mit bzw. ohne (nur Cubecom–Bibliothek MPI–implementiert) Verwendung des Kommunikatorprinzips.

Hierbei wurden die Partitionierungen im Preprocessing mit dem Programmsystem *chaco* [HL95] erzeugt. Interessant war vor allem die Fragestellung, inwiefern sich die verschiedenen Verfahren (lineare Verteilung, spektrale Methoden, Terminal Propagation) als lokale Verfahren zur dynamischen Partitionierung bei adaptiver FEM eignen. Für die Referenzverteilungen wurden in Anlehnung an [SFB-9618] auf die beiden Teatraedervernetzungen *cube768* und *spc3-123* (s. Abbildungen 2.2 und 2.3) zurückgegriffen, die als Beispiele für eine regelmäßige (1. Fall) bzw. eine extrem unregelmäßige Vernetzung (2. Fall) gelten können.

Für beide Referenzvernetzungen ergab sich für die einfache MPI–Variante *Rekursive Spektral–Bisektion* mit nachfolgendem *Terminal Propagation* als günstigste Alternative, unter Verwendung des Kommunikatorprinzips erweist sich eine *globale Nachbesserung (Kernighan Lin)* einer durch *Rekursive Spektral–Oktasektion* erzeugten Verteilung als beste Wahl. Während die regelmäßige Struktur der *cube768*–Vernetzung die hypercube–angepaßte Originalvariante bevorteilte, zeigte sich die neue Version bei der diskontinuierlichen *spc3–123*–Vernetzung überlegen. Dennoch sind beide Partitionierungsverfahren zur dynamischen Verteilung ungeeignet, da der Aufwand für die Berechnung die Effizienz extrem beeinträchtigen würde.

Benötigt wird also ein Verfahren zur Kommunikation, das selbst für weniger angepaßte Verteilungen (z.B. lineare Verteilung) ein effizientes Arbeiten ermöglicht (⇒ 2.3.2.5).

Abbildung 2.2: Beispielvernetzung cube768

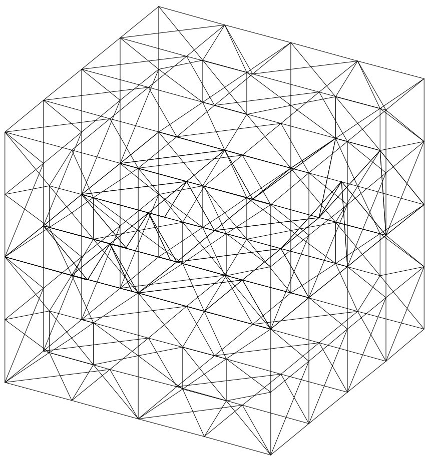
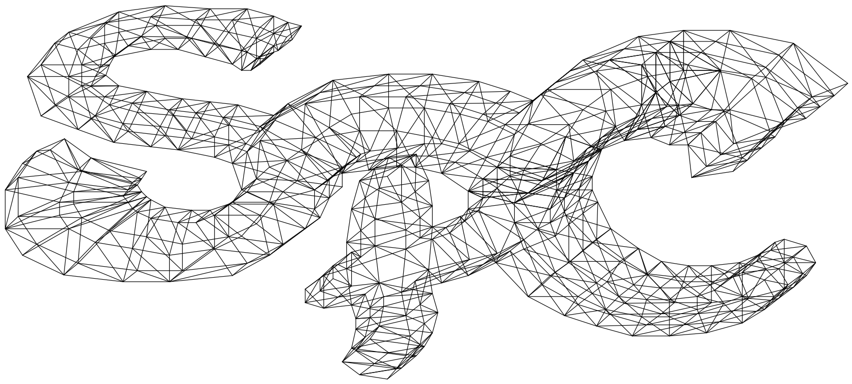


Abbildung 2.3: Beispielvernetzung spc3-123



2.3.2.5 Mappingstrategien für Kommunikatoren

Bei den oben erwähnten Partitionierungsverfahren wurde eine Minimierung des Kommunikationsaufkommens durch eine geschickte Verteilung des betrachteten Gebietes auf die Prozessoren erreicht. Der dazu notwendige Rechenaufwand rechtfertigt allerdings nicht den Einsatz solcher Verfahren zur dynamischen Partitionierung, so wie sie bei adaptiver FEM erforderlich wird.

In [SFB-9810] wird ein Verfahren vorgestellt, das versucht, große Kommunikatorgruppen zu splitten und damit die Anzahl sequentieller Kommunikationsschritte zu verringern. Betrachtet werden hierbei nur solche Kommunikatoren, deren Prozeßanzahl einen Hypercube der Dimension ≥ 2 aufspannen. Existieren Kommunikationsstrukturen zwischen Teilprozessen des Kommunikators, so kann ggf. der Kommunikator auf diese abgebildet werden. Anstatt die Werte eines Teilvektors in einem Schritt zu berechnen, wird dieser jetzt in zwei Abschnitten berechnet, durch geschickte Realisierung des Algorithmus' liegt am Ende des Datenaustauschs das korrekte Ergebnis auf allen beteiligten Prozessen vor. Die erzielte Effizienzsteigerung resultiert aus der geringeren Anzahl an sequentiellen Kommunikationsschritten; es werden nun weniger, dafür etwas längere Datenpakete versendet.

Vergleiche des neuen Verfahrens mit den unter 2.3.2.4 erzeugten Verteilungen bestätigten die Wirksamkeit der Mappingstrategien; das Verfahren erzielte in allen Fällen die besten Ergebnisse.

2.3.3 Teilaufgabe Objektorientierte Realisierung vorliegender (nichtadaptiver) FEM–Systeme

2.3.3.1 Analyse des vorliegenden FEM–Systems

Die 2D–FEM–Bibliothek, die als Grundlage der objektorientierten Realisierung benutzt wird, wurde an der Fakultät für Mathematik von Prof. Dr. Arnd Meyer et.al. entwickelt, und in der Vergangenheit in vielen Anwendungen auch außerhalb der Fakultät für Mathematik eingesetzt. Grundsätzlich besteht diese Bibliothek aus den Modulen Netzgenerator [SFB-9526], Assemblierer und Löser, durch die der Datenstrom geleitet wird. Jedes dieser Module besteht aus einem oder mehreren Library–Dateien, die gemeinsam mit dem der Anwendung entsprechenden Hauptprogramm zu einem statischen Executable gebunden werden. Alle Bestandteile dieser Bibliotheken sind in FORTRAN geschrieben; ein Teil, wurde an spezielle numerische Gegebenheiten auf unterschiedlicher Hardware angepaßt [SPC-9520].

Die einzelnen FORTRAN–Dateien, die sich zu ca. 30000 Zeilen Quelltext summieren, sind sehr sparsam kommentiert, sodaß die Analyse der Algorithmen nicht leicht gefallen ist. Hinzu kommt, daß zwar in bestimmtem Maß Dokumentation verfügbar ist, diese jedoch eher einen Überblick über die Verarbeitung in den entsprechenden Modulen gibt, anstatt tiefer auf verwendete Algorithmen einzugehen.

Probleme, die im Rahmen der Forschungsarbeit zu beheben waren, sind insbesondere:

Statische Speicherverwaltung:

Da FORTRAN nicht über eine dynamische Speicherverwaltung verfügt, wird momentan zu Beginn des Programms ein entsprechend der Hardwareplattform (also dem dort verfügbaren Speicher) und der zu bearbeitenden Problemstellung dimensionierter Speicherbereich durch Anlegen eines großen Bytefeldes “allokiert”. Auf diesem Bytefeld wird dann eine eigene Speicherverwaltung durchgeführt. Wird während der Verfeinerung des FE–Netzes oder während der Assemblierung der Steifigkeitsmatrix die Grenze des verfügbaren Speichers überschritten, so beendet sich das Programm.

Um die statische Größe des Bytefeldes zu ändern, muß eine Datei editiert und das Programm neu übersetzt werden. Wird die Größe von vornherein sehr hoch gewählt, so werden in Abhängigkeit vom zu betrachtenden Problem unter Umständen mehrere Megabyte Speicher verschwendet.

Mangelnde Wart- und Erweiterbarkeit:

Aufgrund der nicht vorhandenen Verknüpfung zwischen Daten und Funktionen im vorliegenden FORTRAN–Quelltext sind bei Erweiterungen stets nicht–codelokale Abhängigkeiten zu betrachten, was zu hohem Aufwand führt. Zusätzlich werden im FORTRAN–Quelltext an verschiedenen Stellen implizite Annahmen über die Zusammensetzung eines eingangs belegten großen

Datenfeldes getroffen, deren Nichteinhaltung bei Veränderungen der Bibliotheken zu schwer auffindbaren Fehlern führt.

2.3.3.2 Anforderungen

Als Schlußfolgerung aus der Analyse der FEM–Bibliotheken und den Zielstellungen der Forschungsarbeit wurde eine Gesamtkonzeption entwickelt, die fortschreitend implementiert und weiterentwickelt wird.

Zusammengefaßt ergeben sich folgende konkretisierte Anforderungen an das objektorientierte FEM-System:

Wart- und Erweiterbarkeit: Durch Verwendung objektorientierter Programmiertechniken wird die Trennung von Daten und Funktionen aufgehoben. Damit wirken sich Veränderungen an Daten oder Funktionen nur noch sehr lokal im Quelltext aus.

Speicherverwaltung: Durch die Verwendung der dynamischen Speicherverwaltung von C++ wird die Inflexibilität der bisher benutzten Speicherverwaltungsmethode überwunden. Dies erhöht die Nutzerfreundlichkeit, da für individuelle Speicherbedürfnisse nicht mehr recompiliert werden muß.

Laufzeiteffizienz: Durch angemessenen Einsatz virtueller Funktionen werden die beim Umstieg von FORTRAN auf C++ wahrscheinlichen Laufzeitverluste minimiert.

Algorithmen: Die von der Fakultät für Mathematik entwickelten numerischen Algorithmen werden auf Portierbarkeit hin untersucht.

2.3.3.3 Objektorientiertes Redesign des FEM–Systems

Das objektorientierte Redesign des in 2.3.3.1 beschriebenen vorliegenden FEM–Systems auf FORTRAN–Basis wurde gemäß den in 2.3.3.2 dargestellten Anforderungen ausgeführt. Kernpunkte des Designs stellen folgende Randbedingungen dar:

- **Es werden nur Konstrukte verwendet, die C++–standardgerecht sind.**
  Die in 2.3 geforderte Portabilität wird allein durch Verwendung eines für die entsprechende Plattform zur Verfügung stehenden standardkonformem C++–Compilers erreicht.

- **Die Verwendung externer Bibliotheken wird weitestgehend vermieden.**
  Damit wird verhindert, daß Abhängigkeiten zu Produkten entstehen, deren Wartung und Weiterentwicklung man selbst nicht in der Hand hat. Selbst wenn es sich dabei um Freeware–Produkte handelt, die prinzipiell selbst weiterentwickelt und gewartet werden können, entstünde so ein erhöhter Aufwand.

Prinzipiell konnte festgestellt werden, daß die Benutzung von Konstrukten der C++–Standardbibliothek für eine Implementierung der derzeitigen FORTRAN–Bibliothek ausreichen, sofern man die algorithmischen Teile der Bibliothek entsprechend nach C++ überträgt. Anderenfalls macht sich die Verwendung einer Numerikbibliothek (z.B. `Blitz++` [B++]) erforderlich.

- **Bei der Implementierung wird die Verwendung bereits dokumentierter Methoden und Algorithmen bevorzugt.**
  Hierbei werden vor allem Methoden und Algorithmen, die in [DP] beschrieben sind, zur Anwendung gebracht. Diese Vorgehensweise erhöht die Wartbarkeit des Systems beträchtlich, da die Konstrukte, ihr Aufbau, mögliche Pre– und Postconditions, sowie Probleme und Abhilfen in der Literatur dokumentiert sind. Darüberhinaus verringert sich auf gleiche Weise der Dokumentationsbedarf.

Unter der Maßgabe dieser Punkte wurde ein zweistufiges Design erarbeitet, das sich in einen funktionalen und einen strukturellen Teil gliedert.

Funktionale Gliederung des FEM–Systems
Grundlage der FEM bildet ein Netzwerk von Knoten, das im Zuge der *Netzgenerierung* verfeinert wird. Anschliessend wird aus dem verfeinerten Netz die Steifigkeitsmatrix *Assembliert*. Diese wird dann im *Löser* weiterverwendet. Der Output des Lösers steht dem Nutzer zur graphischen Darstellung zur Verfügung.

Die funktionalen Teilgebiete der FEM–Bibliothek sind damit *Netzgenerator*, *Assemblierer* und *Löser*. Daraus entstehen die C++–Klassen `Assembler` und `Solver`, die gleichzeitig ein abstraktes Interface darstellen. Damit wird sichergestellt, daß insbesondere verschiedene Lösertypen, die vom gleichen Interface abgeleitet sind, ausgetauscht werden können. Damit werden Erweiterungen der Bibliothek ermöglicht.

Der Netzgenerator wird, obwohl als eigenständige funktionale Komponente ausgewiesen, nicht als selbständige Klasse implementiert. Anstelle dessen erfolgt die Implementierung der Verfeinerungsmethoden direkt in einer Netzwerkklasse `Mesh`, wie im folgenden dargestellt.

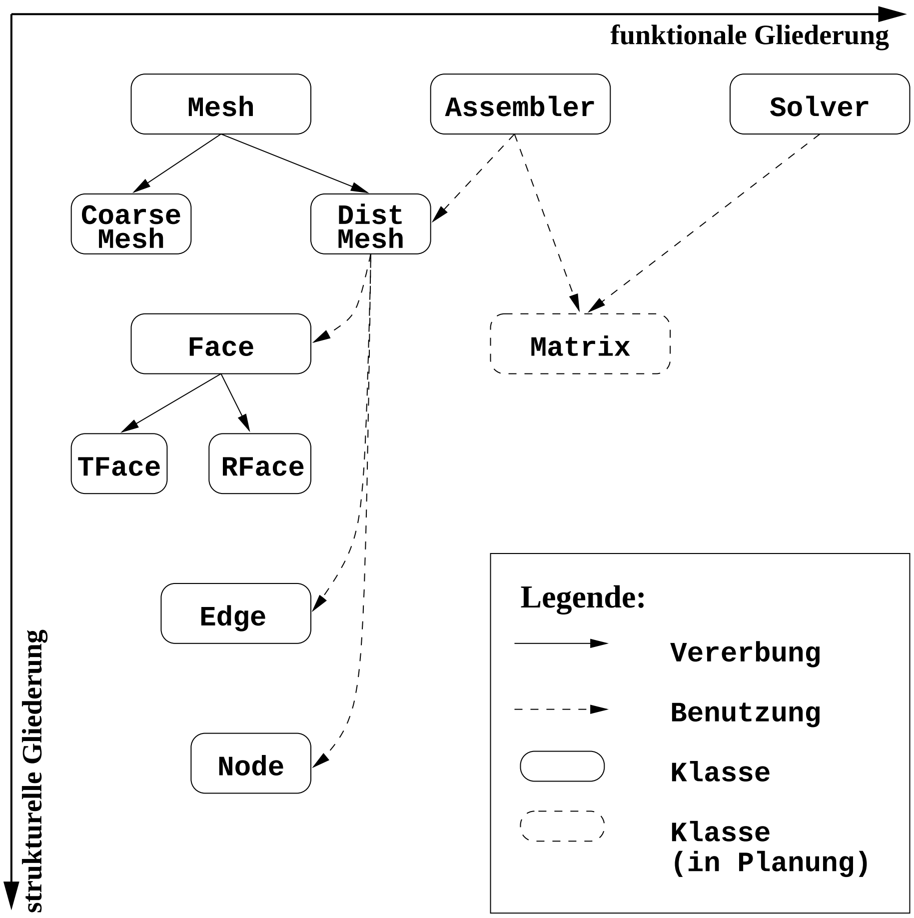
Strukturelle Gliederung des FEM–Systems
Dieser Teil des Designs widerspiegelt die Datenstrukturen, die in dem vorhandenen FEM–System benutzt werden. Als grundlegende Klasse wird hier eine Klasse `Mesh` verwendet, die ein FEM–Netzwerk allgemein repräsentiert. Von dieser Klasse ausgehend werden zwei abgeleitete Klassen `CoarseMesh` – für das vom Nutzer vorgegebene Grobnetz des FEM–Gebietes – und `DistMesh` – für das zum Zwecke der Verfeinerung auf verschiedene Prozessoren verteilte Netz – implementiert. In der Klasse `DistMesh` wird auch die Funktionalität des Netzgenerators übernommen, da die Verfeinerung des FEM–Netzes eine sehr starke logische Bindung an das entsprechende Netz–Objekt hat, und somit als Methode dieses Objektes ausgeführt werden sollte.
In den FEM–Netz–Klassen wiederum werden Instanzen der Klassen `Node`, `Edge`

und `Face` für die Speicherung der relevanten Informationen zu Knoten, Kanten und Flächen verwendet [SFB-9809].

Im weiteren tritt die im Assemblierer aus dem verfeinerten Netz generierte *Steifigkeitsmatrix* auf. Diese wird in der FORTRAN–FEM–Bibliothek durch eine spezielle, in der Fakultät für Mathematik entwickelte, Speicherform als Kompaktliste gespeichert, sodaß die große Anzahl von Nullelementen in dieser Matrix nicht zur Speicherverschwendung führt [SPC-9412]. Eine Arbeitsentscheidung ist, diese Speichermethode objektorientiert umzusetzen. Gegebenenfalls wird diese Entscheidung angesichts des erwarteten Aufwandes revidiert werden müssen, sodaß dann eine externe Numerikbibliothek zum Einsatz kommt. (vgl. aber o.a. Randbedingungen)

Abbildung 2.4 zeigt eine grobe Darstellung der entwickelten Klassenstruktur, welche in einem noch zu veröffentlichenden SFB–Preprint dokumentiert ist.

Abbildung 2.4: Klassenstruktur im OO–Redesign der FEM–Bibliothek



2.3.3.4 Neuimplementierung des Lösers

Ziel der Neuimplementierung des Solvers ist die Widerspiegelung der formalen Algorithmenbeschreibung im C++–Quelltext. Unter Verwendung von Datenstrukturen aus der Bibliothek `libg++` [LG++] erfolgte die Definition verschiedener Utility–Klassen (Matrizen, Vektoren, usw.) und, darauf basierend, eine erste Realisierung des Verfahrens der *Konjugierten Gradienten*.

Die bislang vorliegende Realisierung des Lösers beschränkt sich auf den Basisalgorithmus ohne Vorkonditionierung, die konzeptionellen Grundlagen für das Gesamtverfahren liegen allerdings bereits vor [SFB-9813], [Shanghai97]. In der nachfolgenden Tabelle sollen die Unterschiede zwischen der bislang angewandten Methodik und der vorgesehenen Implementierung kurz skizziert werden. (*Koppelknoten* sind dabei die Crosspoints und alle Knoten auf den Koppelkanten des Grobnetzes; bei *treeup_dod* werden mittels einer kommutativen Operation Daten aller Prozessoren miteinander verknüpft, das Resultat entsteht nur auf Prozessor 0; bei *cube_dod* erhalten dagegen alle Prozessoren das Ergebnis.)

derzeitiges Verfahren	**neues Verfahren**
Prozessor 0 berechnet Grobgittermatrix	alle Prozessoren berechnen Grobgittermatrix
⋮	⋮
Koppelknoten werden mittels tree_dod zu Prozessor 0 gesendet	Koppelknoten werden mittels cube_dod zu allen Prozessoren gesendet
Lösung des Grobgittersystems auf Prozessor 0	Lösung des Grobgittersystems auf allen Prozessoren
Verteilen der Lösung an alle Prozessoren (*tree_down*)	

Unter Verwendung von MPI mit hochoptimierter Allreduce–Implementierung (entspricht cube_dod) ist hier eine deutliche Effizienzsteigerung zu erwarten, auch kann nun bei Einsatz asynchroner Kommunikationsprimitive parallel zur lokalen Grobgittersystem–Lösung die Koppelrandkommunikation angestoßen werden.

2.3.3.5 Implementierung des Netzgenerators Netgen69–C++

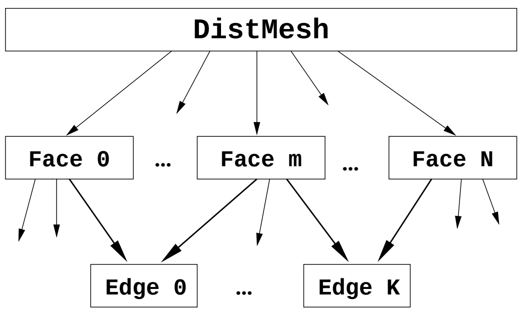
Ausgehend von dem in [SFB-9526] dokumentierten hierarchischen Netzgenerator wurde das in 2.3.3.3 angedeutete Klassenkonzept umgesetzt.

Dabei wurden zunächst die Klassen `Node`, `Edge` und `Face` implementiert, wobei `Face` als Basisklasse für die beiden konkreten Realisierungen `TFace` und `RFace` dient, die speziell die Eigenschaften von Dreiecks– bzw. Vierecksflächen repräsentieren. Diese Klassen bilden das strukturelle Interface, sie widerspiegeln die FEM–typischen Datenstrukturen, wobei alle für diese Datenstrukturen typischen Bearbeitungsfunktionen (wie zum Beispiel `Split` zum Teilen von Kanten und Flächen) gemeinsam mit den verwalteten Daten in jeweils einer Klasse gekapselt sind. Damit wird die spätere Erweiterung erleichtert.

Weiterhin wurde die Klasse `Mesh` implementiert, die aus der strukturellen Betrachtung als Ausgangspunkt der Bearbeitung des vorgegebenen FE–Netzes hervorgeht. Von der abstrakten Basisklasse `Mesh` sind zwei konkrete Klassen `CoarseMesh` und `DistMesh` abgeleitet. In `CoarseMesh` wird ein Grobnetzdatenfile eingelesen. Entsprechend der Anzahl der parallel arbeitenden Prozessoren selektiert jeder einen Anteil an der Gesamtfläche des FE–Netzes und übernimmt die entsprechenden Daten aus den Grobnetzdaten. Daraufhin wird die Verfeinerung des Grobnetzes durch Aufruf der Methode `Face::Split` für alle aus den Grobnetzdaten eingelesenen Flächen des Teilgebietes gestartet. Da jede Fläche Verweise auf die zugehörigen Kanten und jede Kante Verweise auf die ihr angehörenden Knoten enthält, erfolgt die weitere Verarbeitung in den jeweiligen Flächen, sodaß das Hauptprogramm lediglich eine einfache Schleife über alle Flächen enthält.

Die beschriebene Vorgehensweise ist streng objektorientiert und verwirklicht damit die Forderungen betreffend Wartbarkeit und Erweiterbarkeit (s. 2.3.3.2). Außerdem wurden über die normale dynamische Speicherverwaltung von C++ hinausgehende Entwicklungen vorgenommen. So werden die Referenzen der Objekte auf die von ihnen verwalteten Subobjekte über einen Reference Counter verwaltet, um einerseits den gemeinsamen Zugriff von verschiedenen Objekten auf ein beiden angehöriges Objekt zu erlauben, und andererseits die vorzeitige Zerstörung dieses gemeinsam verwalteten Objektes zu verhindern.
In Abbildung 2.5 wird ein solcher gemeinsamer Zugriff dreier Flächenobjekte auf jeweils zwei identische Kantenobjekte dargestellt.

Abbildung 2.5: Zugriff auf gemeinsame Objekte im Netzgenerator



Als Hilfsmittel zur Implementierung dienen Containerklassen der C++ –Standardbibliothek (`vector`, `map`) [C++PL3]. Weiterhin wurde das `Proxy`–Pattern [DP] zur Realisierung des Reference Counters benutzt.

Der neuentstandene objektorientierte hierarchische Netzgenerator Netgen69–C++ ist in [SFB-9809] dokumentiert.

Literaturverzeichnis zu 2.3/(eigene Vorarbeiten und Fremdliteratur)

[C++PL3] Bjarne Stroustrup. *The C++ Programming Language*, 3rd ed. Addison–Wesley, 1997

[DP] Erich Gamma et al. *Design Patterns: elements of reusable object oriented software.* Addison-Wesley, 1997

[B++] `http://monet.uwaterloo.ca/blitz`

[LG++] `http://tinf2.vub.ac.be/~dvermeir/manuals/g++/libg++_toc.html`

[MPI95] *A Message–Passing Interface Standard.* `http://www.mcs.anl.gov/Projects/mpi/index.html`, Juni 1995.

[OOMPI] *Object Oriented MPI (OOMPI): A C++ Class Library for MPI Version 1.0.* `http://www.cse.nd.edu/~lsc/research/oompi/`.

[HL95] B. Hendrickson, R. Leland. *The Chaco User's Guide Version 2.* SAND95–2344. Sandia National Laboratories, Juli 1995.

[SPC-9505] T. Apel, G. Haase, A. Meyer, M.Pester. *Parallel solution of finite element equations systems: efficient inter–processor communication.* SPC–Preprint 5–95. DFG–Forschungsgruppe Scientific Parallel Computing, TU Chemnitz–Zwickau, 1995.

[SPC-9520] G.Haase et al. *Bibliotheken zur Entwicklung paralleler Algorithmen.* SPC–Preprint 95–20. DFG–Forschungsgruppe Scientific Parallel Computing, TU Chemnitz–Zwickau, Juni 1995.

[SFB-9526] C. Israel. *NETGEN69 – Ein hierarchischer paralleler Netzgenerator.* SFB–Preprint 95–26. SFB 393, TU Chemnitz–Zwickau, September 1995.

[SPC-9412] A. Meyer and M. Pester. *Verarbeitung von Sparse–Matrizen in Kompaktspeicherform (KLZ/KZU).* SPC–Preprint 94–12. DFG–Forschungsgruppe Scientific Parallel Computing, TU Chemnitz–Zwickau, Juni 1994.

[SFB-9618] U. Reichel. *Partitionierung von Finite–Elemente–Netzen.* SFB-Preprint 96-18. SFB 393, TU Chemnitz, 1996.

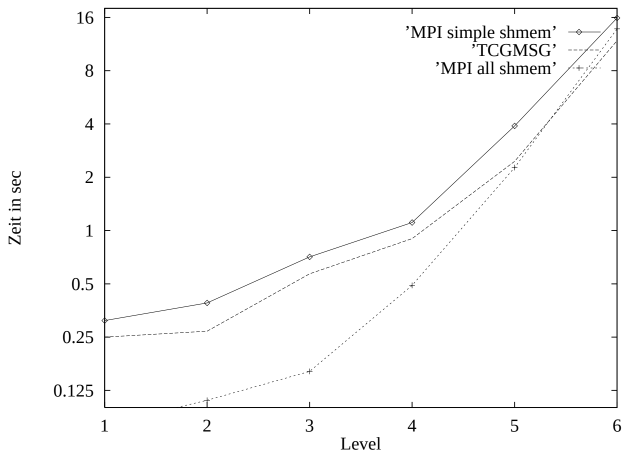
2.4 Ergebnisse

2.4.1 Teilaufgabe Portable Kommunikationsmechanismen für (adaptive) FEM–Systeme

2.4.1.1 MPI als Kommunikationsmechanismus für FEM–Systeme

MPI erweist sich als geeignetes Instrumentarium für alle in FEM–Systemen anfallenden Kommunikationsanforderungen, insbesondere die Ersetzung der Cubecom–Bibliothek durch äquivalente MPI–Routinen verbessert die Effizienz erheblich. Abbildung 2.6 veranschaulicht den Vergleich zwischen der originalen Version (mit Kommunikationsbibliothek TCGMSG), der ersten MPI–Implementierung (MPI simple) und der MPI–Cubecom–Variante (MPI all) im Programmodul *SPC–PM Po2D* unter Verwendung des shared–memory–Devices und der Tetraedervernetzung keule16.

Abbildung 2.6: Rechenzeiten auf SUN4 (4 Prozesse) in Abhängigkeit von der Anzahl der Verfeinerungsschritte.



2.4.1.2 MPI–Kommunikatoren für die Koppelrandkommunikation

Die nachfolgenden Ergebnisse beziehen sich auf Messungen mit dem Programmodul SPC–PM Po3D. Die Aussagen gelten analog für 2D–FEM, jedoch lassen sich in diesem Fall die Resultate auf Grund der relativ kleinen Problemgröße nur geringfügig differenzieren.

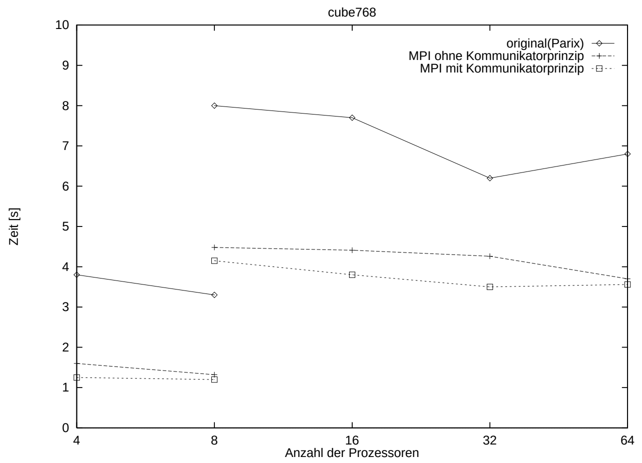
Die Einführung des Kommunikatorprinzips für die Koppelrandkommunikation verbessert das Laufzeitverhalten noch einmal erheblich, vergleicht man die Simulationszeiten mit denen der Parix–optimierten Version des Originalsystems, so wird die Überlegenheit von MPI als Kommunikationsplattform deutlich. Abbildung 2.7 vergleicht die Laufzeiten dieser Varianten, sowie die der besten MPI–Implementierung mit herkömmlicher Koppelrandkommunikation. Gerechnet wurde auf dem PowerGC unter Verwendung von 4 bis 64 Prozessoren, wobei bei Verwendung von 4 und 8 Prozessoren mit 2 und bei 8 bis 64 Prozessoren mit 3 Verfeinerungsschritten gearbeitet wurde.

2.4.1.3 Mappingstrategien für Kommunikatoren

Die folgenden Resultate wurden unter den in Abschnitt 2.4.1.2 genannten Testbedingungen erzielt.

Die Wirkungsweise des Splitalgorithmus' soll anhand einer Tabelle demonstriert werden. Zusammengefaßt wird das Verhältnis der Anzahl der in Frage kommenden Kommunikatoren zu den tatsächlich gesplitteten Kommunikatoren, sowie das resultierende Verhältnis sequentieller Kommunikationsschritte (Linklevel). Als Testvernetzung dient wiederum cube768.

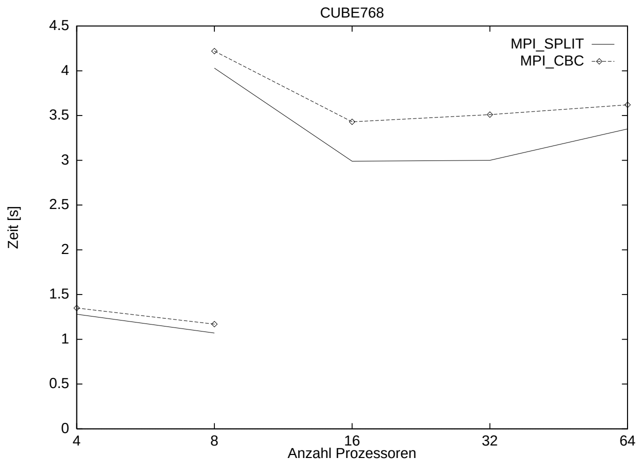
Abbildung 2.7: Rechenzeiten auf PowerGC in Abhängigkeit von der Anzahl der Prozessoren.



Anzahl Prozessoren	**untersuchte Kommunikatoren**	**davon gesplittet**	**Linklevel ohne / mit Splittung**
8	6	6	6 / 3
16	16	16	9 / 5
32	40	31	20 / 16
64	120	58	23 / 23

Abbildung 2.8 zeigt die zugehörigen Simulationszeiten.

Abbildung 2.8: Rechenzeiten auf PowerGC für die Simulation mit und ohne Splittung der Kommunikatoren.



2.4.2 Teilaufgabe Objektorientierte Realisierung vorliegender (nichtadaptiver) FEM–Systeme

2.4.2.1 Der Löser

Durch Verwendung hochoptimierter Standardklassen für Vektoren sind keine Effizienzverluste im Vergleich zur ursprünglichen Implementierung zu verzeichnen, die lokale Skalarproduktbildung geht sogar wesentlich schneller vonstatten, als die originale Routine der FORTRAN–Bibliothek *vbasmod*.

Die globale Berechnung des Skalarproduktes ist als Teil der noch zu implementierenden Klasse *Solver* vorgesehen, sodaß diese ebenso eine Beschleunigung des Verfahrens erbringen sollte.

2.4.2.2 Der Netzgenerator Netgen69–C++

Der objektorientierte hierarchische Netzgenerator Netgen69–C++ ist, wie in 2.3.3.5 dargelegt, bereits im Rahmen des in 2.3.3.3 vorgestellten Redesigns der FEM–Bibliothek entstanden. Seine Implementierung wurde durch zwei im Umfang einer halben Stelle beschäftigte Bearbeiter innerhalb von drei Monaten vorgenommen (daraus resultiert ein rechnerischer Aufwand von 0.25 Mannjahren).

Als Ergebnis kann bereits die Methodik der Implementierung (s. 2.3.3.5) betrachtet werden, da sie die in 2.3.3.3 angedeutete Struktur des objektorientierten FEM–Systems durchsetzt und ihre Implementierbarkeit beweist.

Aus folgender Tabelle kann man den Speicherverbrauch des Netzgenerators im Vergleich zur Anzahl der Verfeinerungsschritte für ein Grobnetzwerk qu128.net entnehmen. Dabei wurde die Messung für einen von vier Prozessoren vorgenommen, der am Anfang 32 der insgesamt 128 Grobnetzflächen zugeordnet bekommt.

Verfeinerungslevel	**Anzahl der Flächen**	**Speicher in kByte**
0	32	
1	128	2012
2	512	2636
3	2048	5148
4	8192	15456
5	32768	56504

Da die Ausstattung des Meßsystems lediglich 32 MB Speicher (+64 MB swap) betrug, konnten weitere Messungen mit höheren Verfeinerungsstufen zu diesem Zeitpunkt nicht erfolgen.

Diese Ergebnisse zeigen den adaptiven Anstieg des Speicherbedarfs in Abhängigkeit von der Anzahl der Verfeinerungslevel. Wäre anstelle des objektorientierten Netzgenerators der des ursprünglichen FORTRAN–FEM–Systems verwendet worden, so hätte im Vorhinein die Größe des verwendeten Speichers festgelegt werden müssen. Für genanntes Meßsystem wäre hier eine Größe des Anfangsfeldes von 20

MB denkbar. Dies hätte dazu geführt, daß bei genanntem Problem eine Verfeinerung in 5 Schritten nicht mehr möglich gewesen wäre, obwohl das Computersystem zur Erfüllung dieser Anforderung durchaus in der Lage ist.

Für weitere Ergebnisse wird auf den Preprint 98-09 ([SFB-9809]) verwiesen.

Literatur zu 2.4/(eigene Veröffentlichungen im Berichtszeitraum)

[RATR-9604] L. Grabowsky, W. Rehm. *Parallel FEM Implementations on Shared Memory Systems.* Computer Architecture Technical Report RA-TR-96-04, TU Chemnitz-Zwickau, 1996

[Shanghai97] L. Grabowsky, W. Rehm. *Efficiency Issues of a Parallel FEM Implementation on Shared Memory Computers*, Proceedings of Advances in Parallel and Distributed Computing Symposium, March 19–21, 1997, Shanghai, PR of China.

[SFB-9809] M. Meyer, T. Ermer. *Der objektorientierte hierarchische Netzgenerator Netgen69–C++.* SFB–Preprint 98–09. SFB 393, TU Chemnitz, 1998

[SFB-9810] T. Ermer. *Mappingstrategien für Kommunikatoren.* SFB–Preprint 98-10. SFB 393, TU Chemnitz, 1998.

[SFB-9813] L. Grabowsky, T. Ermer. *OO–Implementierung eines PPCG–Verfahrens.* SFB–Preprint 98-13. SFB 393, TU Chemnitz, 1998.

[BEY95] U. Beyer. *Portierung eines FEM–Programmes auf Workstation–Cluster mit dem Message Passing Interface (MPI).* Studienarbeit am Lehrstuhl Rechnerarchitektur. TU Chemnitz–Zwickau, Juni 1996.

[ERM96] T. Ermer. *Weiterentwicklung einer MPI–Portierung.* Studienarbeit am Lehrstuhl Rechnerarchitektur. TU Chemnitz, 1996.

[SFB-9717] L. Grabowsky. *MPI–basierte Koppelrandkommunikation und Einfluß der Partitionierung im 3D–Fall.* SFB-Preprint 97-17. SFB 393, TU Chemnitz, 1997.

[RATR-0297] L. Grabowsky, Th. Ermer, J. Werner. *Nutzung von MPI für parallele FEM–Systeme.* Computer Architecture Technical Report RA–TR–97–02, TU Chemnitz, 1997.

[SFB-9722] W. Rehm (Ed.). *Ausgewählte Beiträge zum 1. Workshop Cluster-Computing. 6./7. Nov. 1997.* SFB-Preprint 97-22. SFB 393, TU Chemnitz, 1997.

[Rehm97] W. Rehm (Ed.). *Cluster-Computing. Tagungsband zum 1. Workshop. 6./7. Nov. 1997.* Chemnitzer Informatik–Berichte CSR-97-05, TU Chemnitz (ISSN 0947-5125).

2.5 Offene Fragen/Ausblick

2.5.1 Verfahren zur Kommunikation in FEM–Systemen

Als Fortführung der bisherigen Untersuchungen zu neuen Kommunikationsmechanismen soll auch weiterhin MPI als Basis dienen.

- Mapping von Kommunikatoren ist z.Z. nur auf bestimmte Kommunikatoren (Hypercube u.a. Voraussetzungen, siehe 2.3.2.5 und [SFB-9810]) begrenzt. Kann eine Aufspaltung anderer Kommunikatorgruppen ebenfalls die Kommunikation beschleunigen?
- Die Crosspoint–Berechnung ist noch in der originalen Version implementiert. Ist ein Vorgehen analog zur Koppelrandkommunikation sinnvoll?
- In der neuen OO–Implementierung sind die notwendigen Informationen zu Kommunikationsstrukturen implizit in den Knoten des Netzes vorhanden. Hier stellt sich die Frage, ob sich aus dieser Kenntnis durch rein lokale Verfahren ein global eindeutiger Kommunkationsmechanismus ableiten läßt? (zumindest im 2D–Fall)
- Der bisherige Ansatz zur Koppelrandkommunikation basiert auf den Daten des feinsten Linklevels. Alle relevanten Informationen zur Bildung der Kommunikatoren lassen sich allerdings bereits aus dem Grobnetz gewinnen. Die Realisierung eines derartigen Algorithmus' wird in der objektorientierten Implementierung erfolgen.
- Inwiefern eignet sich OOMPI ([OOMPI]) für die Anforderungen der FEM–Kommunikation?

2.5.2 Weitere OO–Realisierung des FEM–Systems

Zunächst soll durch eine überarbeitete Implementierung von Assemblierer und der Vollendung und Anpassung des Lösers an den STL–Standard auf der Basis des in 2.3.3.3 skizzierten Designs die objektorientierte Realisierung des FEM–Systems verbessert werden. Insbesondere sind Untersuchungen zu weiteren Möglichkeiten der objektorientierten Portierung der numerischen Algorithmen erforderlich, um Aussagen zu erhalten, ob eine befriedigende Modifikation der Algorithmen zur objektorientierten Implementierung noch gefunden werden kann, oder ob bei dem Prinzip der Verwendung externer numerischer Algorithmen letztlich verblieben werden muß.

Die bei der OO–Implementierung des 2D–FEM–Systems gesammelten Erfahrungen können auch in den sich parallel zu diesem Teilprojekt im Teilprojekt A3 entwickelten 3D–FEM–Systemen Anwendung finden.

Parallel zu diesen Arbeiten wird gegenwärtig die Implementierung einer nutzerfreundlichen Schnittstelle zum FEM–System begonnen.