Auslagerung der Ausführung von Methoden der HYPRE Bibliothek in ein Cloudsystem Recherche und Literaturverzeichnis

Thomas Rückert

June 28, 2017

Abstract

abstract formulieren

Contents

List of Figures

List of Tables

Listings

Todo list

1 Einführung in die relevanten Themen

1.1 Allgemeines und Recherche zum Thema Cloud

check das hier:

- nur innerhalb der cloud (ungeeignet): Light-weight remote communication for high-performance cloud networks `http://ieeexplore.ieee.org/document/6483669/`
- Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud `http://ieeexplore.ieee.org/abstract/document/5708447/`

1.1.1 Einführung und Grundbegriffe zu Cloudcomputing

Entwicklung von monolitischen Systemen zu verteilten Anwendungen (SOA, Client/Server). Ermöglicht Auslagerung kostenpflichtiger Berechnungen auf Server, 'schwacher' Client kein Problem mehr. Serverlast kann bei Anwendungen stark schwanken. Zum Beispiel periodische Schwankungen Tag vs Nacht. Klassischer Server muss die hohe Last stemmen, hat dann in anderen Perioden starken leerlauf. Einmalige, sehr hohe Last bei besonderen Situationen (zum Beispiel durch einmalige, nicht wiederkehrende Sportevents wie Olympia/WM). Klassischer Server könnte in diesem Zeitraum komplett ausfallen. Cloud soll dynamische Resource sein, die sich je nach Bedarf skalieren kann.

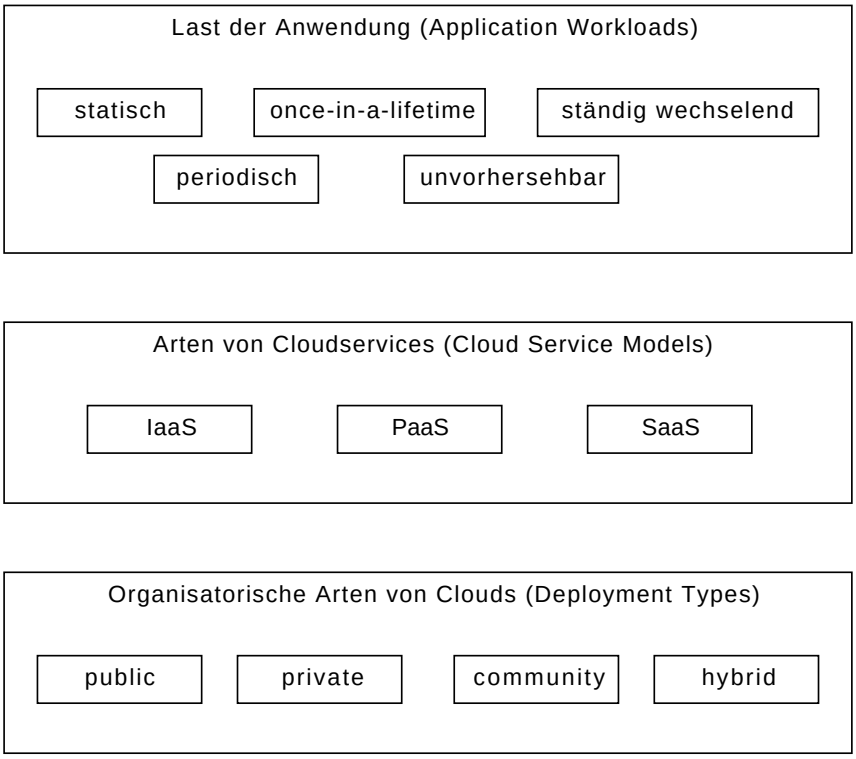
Im Folgenden werden die eben kurz angeschnittenen Eigenschaften von Cloudsystemen näher betrachtet. Ein Übersicht dazu ist in Abbildung 1 gegeben.

Verteilung der Last von Anwendungen Im Buch Cloud Computing Patterns von Christoph Fehling wird die Verteilung von Last in die folgenden Kategorien eingeteilt:

- static workload
- periodic workload
- once-in-a-lifetime workload
- unpredictable workload
- continuously changing workload

Static workload beschreibt eine nicht oder nur minimal schwankende Last. **Periodic workload** hat dagegen wiederkehrende Schwankungen. Diese können zum Beispiel von der Tageszeit abhängig sein. Ein **Once-in-a-lifetime workload** ist eine einmalige Lastspitze. **Unpredictable workload** liegt vor, wenn sich die Last ständig, jedoch unregelmäßig und zufällig verändert, sodass diese nicht vorhersehbar ist. **Continuously changing workload** verändert sich in linear steigend oder fallend.

Figure 1: Übersicht Grundlagen Cloud Computing



1.1.2 NIST definition introduces five fundamental properties that characterize a cloud offering [CloudcomputingPatterns chap. 1.1]

- On-demand self-service
- Broad network access
- Measured service (pay-per-use)
- Resource pooling
- Rapid elasticity

On-demand self-service 'Provisioning' und 'decomissioning' als Aktivitäten zum Hinzufügen oder Entfernen von weiteren Ressourcen. Das kann durch Benutzer über grafische oder Kommandozeilenschnittstellen geschehen oder automatisiert über eine API.

Broad network access Ein starkes Netzwerk ist essentiell um eine Verbesserung durch die Auslagerung von Berechnungen zu erreichen. So kann Zugriffszeit auf Daten weniger abhängig von ihrem pysikalischen Speicherort werden.

genauer definieren

Measured service (pay-per-use) Durch die Nutzung von Cloudsystemen kann man stark von der Flexibilität der Ressourcen profitieren. Diese Flexibilität muss sich auch im Bezahlmodell widerspiegeln.

Resource pooling Ein Cloudsystem benötigt einen (großen) Pool an Ressourcen. Nur so kann Flexibilität für die Nutzer gewährleistet werden. Um eine Austauschbarkeit der Ressourcen zu ermöglichen muss eine homogene Nutzung der Ressourcen existieren.

genauer definieren

warum? flexible Nutzung, Kosten

Rapid elasticity Elastizität von Cloudsystemen ermöglicht eine Effiziente Zuweisung von Ressourcen auf die Nutzer. Der Ressourcepool muss dynamisch unter den Nutzern aufgeteilt werden können.

1.1.3 IDEAL cloud-native applications [CloudcomputingPatterns chap. 1.2]

- Isolated state
- Distribution
- Elasticity
- Automated management
- Loose coupling

Isolated state Cloudanwendungen und ihre Komponenten sollten zustandslos sein. Jede Ressource die zustandslos ist kann deutlich einfacher entfernt oder hinzugefügt werden als eine Ressource mit einem Zustand. So können aufeinander folgende Interaktionen eines Nutzers beliebig auf verschiedene Ressourcen verteilt werden. Eine Ressource die beispielsweise die erste Interaktion getätigt hat wird für weitere Interaktionen nicht mehr benötigt.

Distribution Cloudsysteme können auf viele verschiedene Standorte verteilt sein. In jedem Fall bestehen sie aus vielen verschiedenen Ressourcen. Anwendungen sollten daher aus mehreren Komponenten bestehen, die auf verschiedene Ressourcen verteilt werden können.

Elasticity Horizontale Skalierung statt vertikaler Skalierung: Anwendung soll vom Hinzufügen weiterer Ressourcen profitieren können (horizontal). Es soll nicht nur die 'Verbesserung' einer Ressource eine bessere performance ermöglichen (vertikal). Die Stärke von Cloudsystemen ist die dynamische Zuweisung von Ressourcen. Cloudanwendungen müssen daher horizontal skalieren, also eine Parallelisierbarkeit vorweisen.

Automated management Durch die Elastizität können Ressourcen von Cloudanwendungen während der Laufzeit ständig hinzugefügt und entfernt werden. Diese Aktionen sollten aufgrund von Monitoring der Systemlast ausgelöst werden. Damit die Verwaltung der Ressourcen jederzeit schnell und entsprechend der aktuellen Lage stattfindet sollte sie automatisiert sein.

Loose coupling Da sich die verfügbaren Ressourcen der Anwendung während der Laufzeit ändern können sollten Komponenten möglichst unabhängig voneinander sein. Das reduziert die Fehleranfälligkeit für die Fälle in denen Komponenten kurzzeitig nicht verfügbar sind. Da verteilte Anwendung diese Eigenschaft aufweisen sind Technologien wie Webservices, SOA, asynchrone Kommunikation relevant für Cloudanwendungen.

wie?

Technologien etwas weiter ausführen

1.1.4 Arten von Cloud

Cloud Service Models

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Infrastructure as a Service gibt einem Zugang zu Netzwerk, Computern (unter umständen virtuell) und Speicher. Es ist daher sehr nah an den schon länger existierenden und bekannten Hosted Server Lösungen. **Platform as a Service** dageben entfernt die Notwendigkeit die unterliegende Infrastruktur zu verwalten. Das betrifft Hardware sowie die Betriebssystemebene. Man erhält eine Umgebung in der man seine Anwendung ausführen kann, ohne sich um Themen wie Updates, Kapazitäten von Speicher und anderen Ressourcen oder ähnliche typische Adminaufgaben kümmern zu müssen. **Software as a Service** stellt ein Cloudmodell dar, welches sich eher an Endnutzer richtet. Man erhält

wie zum beispiel, kurz erläutern

Zugriff auf eine Komplette Anwendung, wie zum Beispiel einen Mailserver. Bei dieser Variante muss sich der Nutzer um keinerlei Aufgaben der Verwaltung der Software kümmern.

Organisatorische Arten von Clouds (Deployment Types)

- public
- private
- community
- hybrid

Eine **public cloud** ist für jeden verfügbar. Eine **private cloud** dagegen nur für ein einziges Unternehmen oder einen Nutzer (od. Interessengemeinschaft). Eine **community cloud** liegt zwischen den beiden ersten Varianten. Sie ist in der Regel für eine Menge von Unternehmen verfügbar. Das kann notwendig werden, falls die Unternehmen an einem gemeinsamen Projekt arbeiten. Bei **hybrid clouds** ist von mehreren Clouds die Rede. Diese können aus verschiedenen Arten bestehen und sind untereinander verbunden. So können sich unterschiedliche Anwendungen unter eigenständigen Umgebungen Informationen austauschen und interagieren.

1.1.5 Cloudsysteme

Vergleich in Tabelle? - Konflikt zu chpt. 'Gegenüberstellungen ...'

Verfügbare Cloudsysteme für public Clouds

Amazon EC2 `https://www.youtube.com/watch?v=jLVPqoV4YjU&index=3&list=WL` ssh Zugang siehe ab Minute 50

Instances mit verschiedenen verfügbaren Images = AMI (Linux, Windows). ELB als elastischer Loadbalancer. EBS - elastischer Blockstorage. CloudWatch zum Überwachen von Ressourcen und Applikationen. Kann zum automatischen Skalieren genutzt werden. EC2 Actions - Recover, Stop, Terminate. AWS CodeDeplay erlaubt deploying ohne downtime. Amazon Amazon EC2 Container Service ist ein Containermanagementservice zur Nutzung von Docker Containern.

Kostenloser Testzugang: `https://aws.amazon.com/free/`

Open Source Cloudsystem für private Clouds

- a guide to open source cloud `http://www.tomsitpro.com/articles/open-source-cloud-computing-2-754.html`
- 5 open source cloud platforms `http://solutionsreview.com/cloud-platforms/open-source-cloud-platforms-enterprise/`

Sandstorm Link: https://sandstorm.io/

Kann als private oder public cloud genutzt werden. Kann entweder im bestehenden Cloudsystem von Sandstorm genutzt werden oder selbst gehostet werden. Im Cloudsystem von Sandstorm stehen SaaS und PaaS bereit. Wenn man selbst hostet sind alle Service Modelle verfügbar, also zusätzlich auch IaaS.

Die verfügbaren Platformen stellen ein Linuxsystem bereit. Daher kann jede beliebige Sprache genutzt werden welche auf Linux läuft. Ist aber in erster Linie für SaaS-Apps gedacht.

Fazit: wohl ungeeignet

Openstack Link: https://www.openstack.org/
Openstack kostenlos testen: http://trystack.org/
devstack for getting started with openstack http://docs.openstack.org/developer/devstack/
install openstack on ubuntu https://www.youtube.com/watch?v=jpk4i66-IU4

why openstack? https://www.youtube.com/watch?v=Bk4NoUsikVA

Wird in erster Linie als IaaS verwendet. Man kann bestehende Hardware mit Cloudstack in ein Cloudsystem umwandeln. Es stehen verschiedene Services bereit, mit denen Kernanforderungen wie Speicher (Swift) oder Rechenleistung (Nova) befriedigt werden können. Es gibt darüber hinaus weitere optionale Services für beispielsweise Datenbanken (Trove), Messaging (Zaqar), Container (Magnum) und viele Weitere.

Beispielkonfiguration: https://www.openstack.org/software/sample-configs#high-throughput-computing

GLANCE - Image Service 'Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.'

NOVA - Compute 'Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decomissioning of machines on demand.'

KEYSTONE - Identity 'Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.'

CINDER - Block Storage 'Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.'

Fazit: modulare Services passend für Anforderungen

Apache Cloudstack

Definition

Apache Cloudstack https://cloudstack.apache.org/

'Apache CloudStack is an open source Infrastructure-as-a-Service platform that manages and orchestrates pools of storage, network, and computer resources to build a public or private IaaS compute cloud.

https://cloudstack.apache.org/

With CloudStack you can:
Set up an on-demand elastic cloud computing service. Allow end-users to provision resources' [http://docs.cloudstack.apache.org/en/latest/concepts.html]
private IaaS

1.1.6 Ressourcen

- amazon aws https://aws.amazon.com/types-of-cloud-computing/
- BOOK: cloud computing patterns https://katalog.bibliothek.tu-chemnitz.de/Record/0012763915
- raspberry pi private cloud https://sc5.io/posts/a-private-raspberry-pi-cloud-with-arm-dock
- more raspberry https://www.raspberrypi.org/forums/viewtopic.php?f=36&t=54997

1.1.7 Webtechnologien

RPC (Vergleich zu REST etc)

Service/Webservice SOAP

erklärung service kommunikation allgemein

Socket

erklärung socket allgemein

1.1.8 Mathematische Grundlagen

Laplace-Gleichung Partielle Differentialegleichung. https://www.youtube.com/watch?v=Z9NjA_f2VZw https://www.youtube.com/watch?v=-D4GDdxJrpg

Grundlagen für HYPRE: 2D Laplace

1.2 HYPRE - Überblick über die Bibliothek

HYPRE ist eine freie Software die vom Center for Applied Scientific Computing[3] entwickelt wird. Dieses ist ein Teil der Organisation Lawrence Livermore National Laboratory[4], einer der wichtigsten staatlichen Forschungseinrichtungen der USA[2].

HYPRE ist unter der GNU Lesser General Public License (Free Software Foundation) Version 2.1 lizensiert. Diese ermöglicht die Benutzung der Bibliothek HYPRE sowohl in freier als auch proprietär lizensierter Software. Der Funktionsumfang von HYPRE wird beschrieben als 'Scalable Linear Solvers and Multigrid Methods'.

Die aktuellste Version 2.11 ist seit dem 09.06.2016 verfügbar. Die Bibliothek steht in dieser Version für die Sprachen C (nativ), C++ und FORTRAN bereit. Bis zur vorletzten Minorversion 2.10 wurde HYRPE auch mit dem Babel Interface angeboten. Dieses bot die Möglichkeit HYPRE von anderen Sprachen als C und FORTRAN zu nutzen. So wurde im User Manual die Verwendung aus der Sprache Python beschrieben. Seit Version 2.11.1 steht jedoch keine Implementierung mehr von HYPRE mit Babel Interface zur Verfügung.

HYPRE genauer beschreiben

HYPRE beispiel ex5

1.2.1 Funktionsumfang (und deren Funktionsweise)

Wie bereits erwähnt wird der Funktionsumfang von HYPRE als 'Scalable Linear Solvers and Multigrid Methods' beschrieben.

was heist das? was genau kann man lösen?wie funktioniert das? (lässt sich das beantworten)

Performance Nutzt MPI für Parallelisierbarkeit.

wie löst sie die probleme (groben einblick in die funktionsweise der methoden, falls möglich)

Conceptual Interfaces

Solver Strategies

Preconditioner(s)

performance

1.2.2 Verwendung und Implementierung

Installation Um HYPRE zu installieren sind einige Abhängigkeiten zu erfüllen. Die nötigen Schritte in einem Ubuntu sehen wie folgt aus:

Listing 1: Installation HYPRE in Ubuntu

```
cd /vagrant

//yaourt mpich2
//yaourt openmpi
sudo apt-get install mpich

//Download https://github.com/LLNL/hypre
git clone https://github.com/LLNL/hypre

//Innerhalb des Verzeichnis muss `configure`
//gefolgt von einem `make` ausgef hrt werden.
cd hypre/src

./configure

//Alternativ kann auch das build tool CMake
//eingesetzt werden.
make
```

Implementierung

Checkliste Die folgenden Punkte beschreiben grob die notwendigen Schritte eines Workflows für die Implementierung.

1. Build any necessary auxiliary structures for your chosen conceptual interface. / Datenstrukturen für Gitter (Grid) und Schablone (Stencil) aufbauen, abhängig vom gewählten Interface.

2. Build the matrix, solution vector, and right-hand-side vector through your chosen conceptual interface. / Matrix, Lösungsvektor, right-hand-side Vektor aufbauen. Diese können über verschiedene HYPRE-calls mit den jeweiligen Informationen gefüllt werden.

3. Build solvers and preconditioners and set solver parameters (optional). / Solver und preconditioner aufbauen und deren Parameter setzen. Je nach conceptual interface sind verschiedene solver verfügbar.

4. Call the solve function for the solver. / Solver aufrufen damit das Problem berechnet wird.

5. Retrieve desired information from solver. / Lösung vom solver abrufen.

Zunächst müssen die Datenstrukturen für die In- und Outputdaten definiert werden. Dies können beispielsweise Matrizen oder Vektoren sein. Die Entscheidung über das Inputformat ist abhängig vom gewählten Input-Interface.

Die gewählten Datenstrukturen müssen als nächstes mit den Inputdaten gefüllt werden. HYPRE stellt dafür verschiedene Calls bereit.

beispiele

Der nächste Schritt ist das Anlegen des gewünschten Solvers. Es stehen verschiedene Solver bereit[1], beispielsweise SMG (a parallel semicoarsening multigrid solver) oder BoomerAMG (parallel implementation of the algebraic multigrid method).

1.2.3 Beispielhafter Einblick

In der HYPRE-Bibliothek steht eine Reihe von Beipielen für die Benutzung zur Verfügung. An dieser Stelle wird das Beispiel 5 näher betrachtet. Dieses löst ein zweidimensionales Laplaceproblem (nxn) ohne Randbedingungen. Die Anzahl der Unbekannten beträgt daher N=n. Die komplette Implementierung des Beispiel 5 ist im Anhang zu finden.

Vorbereitungen Da HYPRE für die Parallelisierung auf die MPI Bibliothek zurückgreift muss dieses auch zu Beginn der Anwendung initialsiert werden. Dafür muss durch

```
MPI_INIT()
```

eine grundsätzliche MPI-Session gestartet werden. Weiterhin müssen für die Prozesse ein Kommunikator bestimmt und die eigene id sowie die gesamte Anzahl der Prozesse gelesen werden. Dafür werden die Methoden

```
MPI_Comm_rank()
```

und

```
MPI_Comm_size()
```

aufgerufen.

Listing 2: HYPRE Nutzung: Vorbereitungen

```
//init mpi
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

Matrix anlegen (analog zu Vektor) Matrizen und Vektoren anlegen. Es stehen analoge Methoden für Matrizen und Vektoren bereit. Im folgenden Listing 3 sind die wichtigen Methoden daher lediglich für das Beispiel der Matrix aufgeführt.

Listing 3: HYPRE Nutzung: Matrix anlegen (analog dazu Vektor)

```
//create matrix / vector

HYPRE_IJMatrix A;
int ilower;
int iupper;

//...

/* Create the matrix.
  Note that this is a square matrix, so we indicate the row partition
  size twice (since number of rows = number of cols) */
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, ilower, iupper, ilower, iupper, &A);

/* Choose a parallel csr format storage (see the User's Manual) */
HYPRE_IJMatrixSetObjectType(A, HYPRE_PARCSR);

/* Initialize before setting coefficients */
HYPRE_IJMatrixInitialize(A);

//in einem loop:
HYPRE_IJMatrixSetValues(A, 1, &nnz, &i, cols, values);

/* Assemble after setting the coefficients */
HYPRE_IJMatrixAssemble(A);

/* Get the parcsr matrix object to use */
HYPRE_IJMatrixGetObject(A, (void**) &parcsr_A);
```

Solver anlegen und aufrufen Solver erstellen und aufrufen.

Listing 4: HYPRE Nutzung: Solver anlegen und aufrufen

```
//create solver
//(Boomer)
```

mit Preconditioner Solver mit anderem Preconditioner.

Listing 5: HYPRE Nutzung: Solver mit Preconditioner

```
//create smg (etc) mit anderem precond.
```

1.2.4 Ressourcen

- offiziell: `http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-m`
- Übersicht Publikationen: `http://computation.llnl.gov/projects/hypre-scalable-linear-solve publications`
- Pursuing scalability for hypre's conceptual interfaces `http://dl.acm.org/citation.cfm?doid=1089014.1089018`
- (mehr) beispiele `https://redmine.scorec.rpi.edu/anonsvn/fastmath/docs/ATPESC_2013/Exercises/hypre/examples/README_files/c.html`

1.3 Werkzeuge für die verteilte Ausführung

1.3.1 RPC

rpc aufräumen

RPC steht für Remote Procedure Call. Sie bieten die Möglichkeit von Funktionsaufrufen in verteilten Systemen. Im Gegensatz zu Protokollen wie REST sieht RPC wie ein Methodenaufruf aus. Das Ziel ist die Ausführung einer bestimmten Prozedur, nicht das verwalten einer bestimmten Ressource. Protokolle sind zum Beispiel XML-RPC und Json-RPC.

- understanding rest and rpc for http `https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/`
- Implementing remote procedure calls `http://dl.acm.org/citation.cfm?id=357392`
- c++ json-rpc lib `https://github.com/cinemast/libjson-rpc-cpp/`

REST vs RPC

skizze rpc call (vs skizze rest call)

warum rest ungeeignet

XML-RPC XML-RPC ist ein Protokoll für Remote Procedure Calls. In diesem wird XML zur Formatierung der übertragenen Daten verwendet. Wie in RPC besteht auch in XML-RPC der Bezug auf Methoden. Darin besteht der Unterschied zu anderen Protokollen, welche sich auf die Manipulation von Daten beziehungsweise Ressourcen beziehen. Es gibt in verschiedenen Sprachen Implementierungen von Server-Client-Frameworks die das XML-RPC Protokoll zur Kommunikation verwenden.

xml-rpc XML-RPC Bibliothek für C und C++ `http://xmlrpc-c.sourceforge.net/`.

Installation:

Listing 6: XML-RPC Installation

```
./configure
make
make install
//And then, if Linux:
ldconfig
```

Diese Bibliothek bietet ein performantes Framework zur Implementierung von XML-RPC Anwendungen. Sie bietet sicher Kommunikation per HTTPS. Es können größere Datenmengen per Packet Stream übertragen werden. Die Verwendung der Frameworkmethoden erfolgt sehr transparent und modular, wodurch man leicht Einfluss auf die funktionsweise nehmen kann. Das führt allerdings auch dazu, dass im Vergleich zu modernen Frameworks recht viel boiler plate code erforderlich ist.

JSON-RPC JSON-RPC ist wie XML-RPC ein Protokoll für Remote Procedure Calls. Dieses verwendet für die Formatierung der Daten das Json-Format, was den wesentlichen Unterschied zum XML-RPC darstellt. Das Json-Format benötigt im Vergleich zu XML wesentlich weniger Zeichen für die Strukturierung der Daten. Durch die unterschiedliche Effizienz in der Repräsentation der Daten ergeben sich bei JSON-RPC bessere Übertragungsgeschwindigkeiten. Die Auswahl an JSON Bibliotheken welche in C und C++ implementiert sind ist groß. Im folgenden github-Projekt wurde eine große Auswahl miteinander verglichen: `https://github.com/miloyip/nativejson-benchmark`.

liste durchgehen

- information: `http://www.simple-is-better.org/json-rpc/`
- wiki: `https://en.wikipedia.org/wiki/JSON-RPC`
- json rpc server: `https://github.com/hmng/jsonrpc-c`
- tcp client: `http://jsonrpc-cpp.sourceforge.net/index.php?n=Main.HomePage`
- rpc client?: `https://groups.google.com/forum/#!topic/json-rpc/9O1dbQehU04`

Im folgenden Abschnitt wird eine Auswahl von JSON-RPC Bibliotheken vorgestellt:

json-rpc json-rpc-libs:

- `https://github.com/yeryomin/libjrpc`
- `https://github.com/jhlee4bb/jsonrpC`
- `https://github.com/hmng/jsonrpc-c`
- `https://github.com/pijyoi/jsonrpc`

yeryomin/libjrpc `https://github.com/yeryomin/libjrpc`
install:

Listing 7: yeryomin/libjrpc Installation

```
git clone git@github.com:yeryomin/libjrpc.git
git clone git@github.com:yeryomin/libipsc.git
git clone git@github.com:yeryomin/libfmt.git
git clone git@github.com:zserge/jsmn.git
git clone git@github.com:yeryomin/liba.git

cd libipsc
make
sudo ln -s /path/to/libipsc.h /usr/include
cd ..

cd jsmn
make
sudo ln -s /path/to/jsmn.h /usr/include
cd ..

cd libfmt
make
sudo ln -s /path/to/libfmt.h /usr/include
cd ..

cd libjrpc
make
sudo ln -s /path/to/libjrpc.h /usr/include
```

so many unresolved dependencies..... wow

jhlee4bb/jsonrpC (needs libwebsockets installed: ‘yaourt libwebsockets‘ etc.)

Listing 8: jhlee4bb/jsonrpC Installation

```
git clone git@github.com:jhlee4bb/jsonrpC.git
cd jsonrpC
mkdir build
cd build
cmake ..
make
build output left in jsonrpc-x.y
```

völlig veraltet - websocket-lib ist inziwschen komplett inkompatibel

Bibliotheken/Frameworks/Implementierungen

client and server stubs: https://github.com/masroorhasan/RPC

rpc for php and c: https://github.com/laruence/yar

Mercury Homepage: https://mercury-hpc.github.io/ Dokumentation: http://mercury-hpc.github.io/documentation/ High-level RPC Layer: http://mercury-hpc.github.io/documentation/#high-level-rpc-layer Mailing-List: https://lists.mcs.anl.gov/mailman/private/mercury/

Für HPC optimierte c-Bibliothek. Bietet außerdem MPI-Unterstützung. (Etwas) Dokumentation vorhanden.

Trios/Nessie https://software.sandia.gov/trac/nessie/wiki/WikiStart

Portals https://www.researchgate.net/publication/221201996_Efficient_Data-Movement_for_Lightweight_IO

DART http://coewww.rutgers.edu/www4/cacweb/TASSL/Papers/dart_hpdc.pdf

1.3.2 andere

Python in C Ausführen

ressourcen https://docs.python.org/2.5/ext/callingPython.html https://www.codeproject.com/articles/11805/embedding-python-in-c-c-part-i

better

http://www.linuxjournal.com/article/8497 https://www.codeproject.com/articles/820116/embedding-python-program-in-a-c-cplusplus-code

basics some text

Listing 9: Python in C ausführen

```
#include <python3.6m/Python.h>

int main()
{
  Py_Initialize();
  PyRun_SimpleString(
    "print('Hello World from Embedded Python.')"
  );
  Py_Finalize();
}
```

kompilieren von c-code mit python3.6

Listing 10: Python in C kompilieren

```
cc prog.c -o prog.o -I/usr/include/python3.6m -lpython3.6m
  -lm -L/usr/lib/python3.6
```

spring-server framework https://github.com/bartobri/spring-server

Listing 11: Installation bartobri/spring-server

```
git clone git@github.com:bartobri/spring-server.git
```

1.3.3 Serialisierung

https://www.quora.com/Is-there-a-C-struct-to-JSON-generator-library

Protocol Buffers

protobuf überarbeiten und nach oben zu grpc verweisen

protobuf: https://github.com/protobuf-c/protobuf-c-rpc

https://developers.google.com/protocol-buffers/ Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data. Implementierung für C: https://github.com/protobuf-c/protobuf-c https://github.com/protobuf-c/protobuf-c-rpc

Fazit Generiert den Code für structs selbstständig. Für den Anwendungsfall bestehen die Strukturen jedoch bereits und müssen umgewandelt werden. Ist daher ungeeignet.

2 Gegenüberstellung verfügbarer Werkzeuge und Varianten

vergleiche kapitel ausarbeiten

2.1 'private' und 'public' Cloud

- gibt es open source cloud systeme? welche?
- welche public clouds gibt es (zB aws, windows azure, adobe creative)
- was sind unterschiede (neben dem access, zB performance?)

2.2 Technologie

tabelle/vergleiche zwischen technologien: rpc,socket,service etc

- welche Werkzeuge für die kommunikation zwischen client und cloud
- rpc, socket, service ...
- abwägen zwischen performance, aufwand ...
- welche framework könnten genutzt werden

2.3 Cloudtyp

wird evtl schon teilweise in abschnitt 1 (allgemeines zur cloud) abgedeckt

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)
- aber entscheidend: wie eignen sich diese typen für 'unsere' Implementierung

2.4 Sprache

- hardware-nah: C/C++ (bessere performance)
- vs netzwerknah: (bessere möglichkeiten die kommunikation zu Implementieren)
- kann eine hybridform eingsetzt werden? zB (micro-)service: bib in c kommuniziert mit client-backend in php, dieses führt die calls zum server aus

2.5 Vergleich ausgewählter Cloudsysteme in einer Tabelle

anhand relevanter (für das Projekt) Eigenschaften

Table 1: Vergleich von Cloudanbietern

Cloudsystem	private	public	PaaS	...	...
Openstack	☑	...	...	...	...
Apache CS	☑	...	...	...	...
Amazon AWS	☐	...	☑	...	...

3 Auswertung der Vergleiche und Auswahl der Werkzeuge

Vergleiche verschiedener Technologien und Werkzeuge. Auswahl für den Einsatz bei der Implementierung.

3.1 gRPC

`http://www.grpc.io/`
Googles open source RPC framework. Dokumentation: `http://www.grpc.io/docs/`.

Installation Installiere vagrant ubuntu 16.04. Setup von gRPC nach der Installation:

Listing 12: Installiere vagrant ubuntu 16.04

```
cd /vagrant
sudo apt-get install build-essential autoconf libtool
sudo apt-get install libgflags-dev libgtest-dev
sudo apt-get install clang libc++-dev
git clone -b $(curl -L http://grpc.io/release) \\
https://github.com/grpc/grpc
cd grpc
git submodule update --init
make && sudo make install
cd third_party/protobuf
sudo apt-get install unzip
./autogen.sh
./configure
make && sudo make install
cd -
cd examples/cpp/helloworld/
sudo apt-get install pkg-config
make
```

kurze Einführung grpc genauer beschreiben (auch protobuf)

Benutzt Protobuf zum erstellen von Services und Nachrichtenobjekten (stubs). Offizieller support für C++ vorhanden. Implementierung für c vorhanden `https://github.com/protobuf-c/protobuf-c`. Das Basisframework von gRPC ist

ebenfalls in nativem C geschrieben. Die Highlevel-Framework-API ist jedoch nicht in C verfügbar.

Da die Definitionen der stubs nicht sprachgebunden erfolgen ist eine (spätere) Umstellung auf eine andere Sprache einfacher. Es müssen die stubs für die neue Sprache lediglich aus den gleichen Definitionen neu generiert werden.

Listing 13: Beispiel grpc

```
package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Überblick Funktionsumfang Ein definierter Service steht nach dem generieren der Implementierung in der gewünschten Sprache als Interface bereit. Dieses kann alternativ auch als AsyncService implementiert werden.

Listing 14: grpc Service Definition

```
//synchron
class X : public Greeter::Service {}
//oder asynchron
class X : public Greeter::AsyncService {}
```

Listen von Request- oder Responseobjekten können in Form eines Streams übertragen werden.

Listing 15: grpc Funktion Definition

```
rpc ListFeatures(Rectangle) returns (stream Feature) {}
```

3.2 Ressourcen

- HPC in der Cloud `http://grids.ucs.indiana.edu/ptliupages/publications/cloud_handbook_final-with-diagrams.pdf`

3.3 Roadmap Implementierung

roadmap checken/erweitern/abhaken

Was muss zwischen Client und Server übertragen werden: Anzahl der calls zwischen Client und Server sollte so gering wie möglich gehalten werden. Der Aufbau der Datenstrukturen sollte daher komplett innerhalb des Clients passieren. Sobald alle Eingabedaten erstellt sind muss ein Solver aufgerufen werden. Die Solver sollten auf dem Server ausgeführt werden.

(später:) Eventuell kann sogar der Aufwand der Berechnung abgeschätzt werden. Somit könnten einfache Berechnungen innerhalb des Client ausgeführt werden. Lediglich komplexe Berechnungen an den Server übermittelt werden.

Fazit 1. Möglichkeit: Es müssen (alle) HYPRE-Datentypen übertragen werden können. Dazu muss ein Mapper existieren. Dieser sollte die Datenstrukturen in ein Datenformat umwandeln welches übertragen werden kann (json). Er muss außerdem die übertragbaren Daten wieder in die Datenstrukturen zurück umwandeln können.

2. Möglichkeit: Alle Eingabeparameter und Funktionsaufrufe werden gesammelt. Auf Seite des Servers werden diese nacheinander ausgeführt um die Datenstrukturen zu bilden. Danach wird der Solver ausgeführt. Am Ende die lediglich die Antwort umgewandelt und übertragen.

Client

Anforderungen Bibliothek. Alle HYPRE-Calls implementieren die einen solver ausführen. Statt der direkten Ausführung müssen die Input-Daten an den Server übertragen werden. Außerdem muss der Call-Name übertragen werden. Nach Ausführung auf dem Server muss die Antwort wiederum in die entsprechenden Datentypen umgewandelt werden.

Kernmodule Einige HYPRE-Methoden überschreiben. Mapping HYPRE-Datentypen zu JSON. Mapping JSON zu HYPRE-Datentypen. Webservices implementieren für Komunikation mit dem Server.

4 Beispielinstallation OpenStack

http://docs.openstack.org/developer/devstack/
https://www.youtube.com/watch?v=jpk4i66-IU4
http://ronaldbradford.com/blog/setting-up-ubuntu-on-virtualbox-for-devstack-2016-03-30/
http://ronaldbradford.com/blog/setting-up-ubuntu-using-vagrant-2016-04-01/
http://ronaldbradford.com/blog/downloading-and-installing-devstack-2016-04-02/

Automatisches Setup: https://github.com/lorin/devstack-vm

4.1 Vorbereitung für die Installation

Ubuntu 16.04 in Vagrant aufsetzen

1. `vagrant init bento/ubuntu-16.04`
2. Vagrantfile anpassen. Es sollten mindestens 4GB RAM zugewiesen werden, damit OpenStack performant läuft.

Listing 16: Installation Ubuntu 16.04 in Vagrant

```
Vagrant.configure(2) do |config|
  config.vm.box = "bento/ubuntu-16.04"
  config.vm.network "private_network", type: "dhcp"

  config.vm.provider "virtualbox" do |v|
    v.memory = 4096
  end
end
```

3. `vagrant up`
4. `vagrant ssh`

4.2 Installation

Devstack installieren

1. `git clone https://git.openstack.org/openstack-dev/devstack`
2. `cd devstack`
3. Bei Bedarf zu einem (älteren) stable release wechseln:

   `git checkout stable/ocata`

   oder

   `git checkout stable/mitaka`

   Die master-branch kann auch genutzt werden.
4. `ifconfig enp0s8 | grep addr`

5. `inet addr kopieren`
6. `cp samples/local.conf .`
7. `HOST_IP="xxx.xxx.xxx.xxx"`
8. `echo "HOST_IP=${HOST_IP}" >> local.conf`
9. Es muss ein user mit sudo-rechten (ohne passwort) existieren, der nicht root ist. Der Standarduser vagrant ist dafür geeignet.
10. `./stack.sh`

Am Ende der erfolgreichen Installation erscheint die folgende Ausgabe in etwa so. Von hier können die nötigen Informationen für das weitere Vorgehen entnommen werden.

Listing 17: Ausgabe von DevStack

```
=========================
DevStack Component Timing
=========================
Total runtime          3644

run_process              87
test_with_retry           7
apt-get-update           12
pip_install             881
restart_apache_server    20
wait_for_service         61
git_timed               355
apt-get                 457
=========================



This is your host IP address: 172.28.128.3
This is your host IPv6 address: ::1
Horizon is now available at http://172.28.128.3/dashboard
Keystone is serving at http://172.28.128.3/identity/
The default users are: admin and demo
The password: nomoresecret
```

4.3 Starten

Openstack (Horizon) starten Die folgenden Informationen sind abhängig von der Konfiguration. Für dieses Beispiel sind folgende Daten notwendig.

1. `http://172.28.128.3/dashboard` in einem Browser öffnen
2. Benutzername admin oder demo
3. Password nomoresecret

Reboot `https://ask.openstack.org/en/question/5423/rebooting-with-devstack/`
Wenn die Vagrantbox herunter gefahren wurde muss devstack nach einem erneuten Start ebenfalls neu gestartet werden. Ansonsten können die verschiedenen Services nicht genutzt werden. Das Webfrontend Horizon kann in diesem Fall keinerlei Ressourcen laden. Zum erneuten Erstellen der openstack Umgebung können die folgenden Befehle genutzt werden.

```
./unstack.sh
./stack.sh
```

Bei einem erneuten Ausführen von stack.sh würden jedoch alle Datenbanken neu erstellt werden. Das würde zu einem kompletten Datenverlust führen. Dies ist aber nicht in jedem Fall gewünscht. Dafür steht eine screen-Konfiguration bereit mit der die notwendigen Services gestartet werden können.
`http://stackoverflow.com/questions/36268822/no-rejoin-stack-sh-script-in-my-setup`

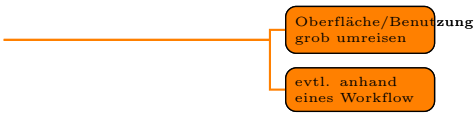
```
screen -c stack-screenrc
```

So kann eine bestehende openstack-Konfiguration auch nach einem Neustart weiter genutzt werden.

4.4 Benutzung Beispielhaft

Verbindung zur Cloud http://openstack-cloud-mylearning.blogspot.de/2015/02/openstack-how-to-access-vm-using-ssh.html

Überblick Oberfläche und Funktionen erläutern



5 Zukünftige, weiterführende Arbeiten

eventuell werden Teile von hier in die aktuelle Arbeit verschoben

- Skalierbarkeit der Cloud (einsetzen)
- Vorteile der Auslagerung:
  - performance
  - speicher
  - lösbarkeit (nur remote überhaupt lösbar)

References

[1] Center for Applied Scientific Computing (Lawrence Livermore National Laboratory). *User's Manual HYPRE*, 03 2017.

[2] Center for Applied Scientific Computing (Lawrence Livermore National Laboratory). Hypre software releases. zuletzt besucht am 28.06.2017.

[3] Center for Applied Scientific Computing (Lawrence Livermore National Laboratory). Überlick center for applied scientific computing. zuletzt besucht am 28.06.2017.

[4] Lawrence Livermore National Laboratory. Überblick lawrence livermore national laboratory. zuletzt besucht am 28.06.2017.

A
HYPRE

A.1 HYPRE Example 5

```
/*
   Example 5

   Interface:    Linear-Algebraic (IJ)

   Compile with: make ex5

   Sample run:   mpirun -np 4 ex5

   Description:  This example solves the 2-D Laplacian problem with zero bounda
                 conditions on an n x n grid.  The number of unknowns is N=n^2.
                 The standard 5-point stencil is used, and we solve for the
                 interior nodes only.

                 This example solves the same problem as Example 3.
Available
                 solvers are AMG, PCG, and PCG with AMG or Parasails
                 preconditioners.  */

#include <math.h>
#include "_hypre_utilities.h"
#include "HYPRE_krylov.h"
#include "HYPRE.h"
#include "HYPRE_parcsr_ls.h"

#include "vis.c"

int hypre_FlexGMRESModifyPCAMGExample(void *precond_data, int iterations,
                                      double rel_residual_norm);


int main (int argc, char *argv[])
{
   int i;
   int myid, num_procs;
   int N, n;

   int ilower, iupper;
   int local_size, extra;

   int solver_id;
   int vis, print_system;

   double h, h2;
```

```
HYPRE_IJMatrix A;
HYPRE_ParCSRMatrix parcsr_A;
HYPRE_IJVector b;
HYPRE_ParVector par_b;
HYPRE_IJVector x;
HYPRE_ParVector par_x;

HYPRE_Solver solver, precond;

/* Initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

/* Default problem parameters */
n = 33;
solver_id = 0;
vis = 0;
print_system = 0;


/* Parse command line */
{
   int arg_index = 0;
   int print_usage = 0;

   while (arg_index < argc)
   {
      if ( strcmp(argv[arg_index], "-n") == 0 )
      {
         arg_index++;
         n = atoi(argv[arg_index++]);
      }
      else if ( strcmp(argv[arg_index], "-solver") == 0 )
      {
         arg_index++;
         solver_id = atoi(argv[arg_index++]);
      }
      else if ( strcmp(argv[arg_index], "-vis") == 0 )
      {
         arg_index++;
         vis = 1;
      }
      else if ( strcmp(argv[arg_index], "-print_system") == 0 )
      {
         arg_index++;
         print_system = 1;
      }
      else if ( strcmp(argv[arg_index], "-help") == 0 )
```

```
      {
         print_usage = 1;
         break;
      }
      else
      {
         arg_index++;
      }
   }

   if ((print_usage) && (myid == 0))
   {
      printf("\n");
      printf("Usage: %s [<options>]\n", argv[0]);
      printf("\n");
      printf("  -n <n>              : problem size in each direction (defaul
      printf("  -solver <ID>        : solver ID\n");
      printf("                        0  - AMG (default) \n");
      printf("                        1  - AMG-PCG\n");
      printf("                        8  - ParaSails-PCG\n");
      printf("                        50 - PCG\n");
      printf("                        61 - AMG-FlexGMRES\n");
      printf("  -vis                : save the solution for GLVis visualizat
      printf("  -print_system       : print the matrix and rhs\n");
      printf("\n");
   }

   if (print_usage)
   {
      MPI_Finalize();
      return (0);
   }
}

/* Preliminaries: want at least one processor per row */
if (n*n < num_procs) n = sqrt(num_procs) + 1;
N = n*n; /* global number of rows */
h = 1.0/(n+1); /* mesh size*/
h2 = h*h;

/* Each processor knows only of its own rows - the range is denoted by ilowe
   and upper.  Here we partition the rows. We account for the fact that
   N may not divide evenly by the number of processors. */
local_size = N/num_procs;
extra = N - local_size*num_procs;

ilower = local_size*myid;
ilower += hypre_min(myid, extra);

iupper = local_size*(myid+1);
```

```
iupper += hypre_min(myid+1, extra);
iupper = iupper - 1;

/* How many rows do I have? */
local_size = iupper - ilower + 1;

/* Create the matrix.
   Note that this is a square matrix, so we indicate the row partition
   size twice (since number of rows = number of cols) */
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, ilower, iupper, ilower, iupper, &A);

/* Choose a parallel csr format storage (see the User's Manual) */
HYPRE_IJMatrixSetObjectType(A, HYPRE_PARCSR);

/* Initialize before setting coefficients */
HYPRE_IJMatrixInitialize(A);

/* Now go through my local rows and set the matrix entries.
   Each row has at most 5 entries. For example, if n=3:

   A = [M -I  0; -I M -I; 0 -I M]
   M = [4 -1  0; -1 4 -1; 0 -1 4]

   Note that here we are setting one row at a time, though
   one could set all the rows together (see the User's Manual).
*/
{
   int nnz;
   double values[5];
   int cols[5];

   for (i = ilower; i <= iupper; i++)
   {
      nnz = 0;

      /* The left identity block:position i-n */
      if ((i-n)>=0)
      {
         cols[nnz] = i-n;
         values[nnz] = -1.0;
         nnz++;
      }

      /* The left -1: position i-1 */
      if (i%n)
      {
         cols[nnz] = i-1;
         values[nnz] = -1.0;
         nnz++;
      }
```

```
      /* Set the diagonal: position i */
      cols[nnz] = i;
      values[nnz] = 4.0;
      nnz++;

      /* The right -1: position i+1 */
      if ((i+1)%n)
      {
         cols[nnz] = i+1;
         values[nnz] = -1.0;
         nnz++;
      }

      /* The right identity block:position i+n */
      if ((i+n)< N)
      {
         cols[nnz] = i+n;
         values[nnz] = -1.0;
         nnz++;
      }

      /* Set the values for row i */
      HYPRE_IJMatrixSetValues(A, 1, &nnz, &i, cols, values);
   }
}

/* Assemble after setting the coefficients */
HYPRE_IJMatrixAssemble(A);

/* Note: for the testing of small problems, one may wish to read
   in a matrix in IJ format (for the format, see the output files
   from the -print_system option).
   In this case, one would use the following routine:
   HYPRE_IJMatrixRead( <filename>, MPI_COMM_WORLD,
                       HYPRE_PARCSR, &A );
   <filename> = IJ.A.out to read in what has been printed out
   by -print_system (processor numbers are omitted).
   A call to HYPRE_IJMatrixRead is an *alternative* to the
   following sequence of HYPRE_IJMatrix calls:
   Create, SetObjectType, Initialize, SetValues, and Assemble
*/


/* Get the parcsr matrix object to use */
HYPRE_IJMatrixGetObject(A, (void**) &parcsr_A);


/* Create the rhs and solution */
HYPRE_IJVectorCreate(MPI_COMM_WORLD, ilower, iupper,&b);
```

```
HYPRE_IJVectorSetObjectType(b, HYPRE_PARCSR);
HYPRE_IJVectorInitialize(b);

HYPRE_IJVectorCreate(MPI_COMM_WORLD, ilower, iupper,&x);
HYPRE_IJVectorSetObjectType(x, HYPRE_PARCSR);
HYPRE_IJVectorInitialize(x);

/* Set the rhs values to h^2 and the solution to zero */
{
   double *rhs_values, *x_values;
   int    *rows;

   rhs_values = (double*) calloc(local_size, sizeof(double));
   x_values = (double*) calloc(local_size, sizeof(double));
   rows = (int*) calloc(local_size, sizeof(int));

   for (i=0; i<local_size; i++)
   {
      rhs_values[i] = h2;
      x_values[i] = 0.0;
      rows[i] = ilower + i;
   }

   HYPRE_IJVectorSetValues(b, local_size, rows, rhs_values);
   HYPRE_IJVectorSetValues(x, local_size, rows, x_values);

   free(x_values);
   free(rhs_values);
   free(rows);
}


HYPRE_IJVectorAssemble(b);
/*  As with the matrix, for testing purposes, one may wish to read in a rhs:
    HYPRE_IJVectorRead( <filename>, MPI_COMM_WORLD,
                                 HYPRE_PARCSR, &b );
    as an alternative to the
    following sequence of HYPRE_IJVectors calls:
    Create, SetObjectType, Initialize, SetValues, and Assemble
*/
HYPRE_IJVectorGetObject(b, (void **) &par_b);

HYPRE_IJVectorAssemble(x);
HYPRE_IJVectorGetObject(x, (void **) &par_x);


/*  Print out the system  - files names will be IJ.out.A.XXXXX
    and IJ.out.b.XXXXX, where XXXXX = processor id */
if (print_system)
{
```

```
   HYPRE_IJMatrixPrint(A, "IJ.out.A");
   HYPRE_IJVectorPrint(b, "IJ.out.b");
}


/* Choose a solver and solve the system */

/* AMG */
if (solver_id == 0)
{
   int num_iterations;
   double final_res_norm;

   /* Create solver */
   HYPRE_BoomerAMGCreate(&solver);

   /* Set some parameters (See Reference Manual for more parameters) */
   HYPRE_BoomerAMGSetPrintLevel(solver, 3);  /* print solve info + parameter
   HYPRE_BoomerAMGSetOldDefault(solver); /* Falgout coarsening with modified
   HYPRE_BoomerAMGSetRelaxType(solver, 3);   /* G-S/Jacobi hybrid relaxation
   HYPRE_BoomerAMGSetRelaxOrder(solver, 1);   /* uses C/F relaxation */
   HYPRE_BoomerAMGSetNumSweeps(solver, 1);   /* Sweeeps on each level */
   HYPRE_BoomerAMGSetMaxLevels(solver, 20);  /* maximum number of levels */
   HYPRE_BoomerAMGSetTol(solver, 1e-7);      /* conv. tolerance */

   /* Now setup and solve! */
   HYPRE_BoomerAMGSetup(solver, parcsr_A, par_b, par_x);
   HYPRE_BoomerAMGSolve(solver, parcsr_A, par_b, par_x);

   /* Run info - needed logging turned on */
   HYPRE_BoomerAMGGetNumIterations(solver, &num_iterations);
   HYPRE_BoomerAMGGetFinalRelativeResidualNorm(solver, &final_res_norm);
   if (myid == 0)
   {
      printf("\n");
      printf("Iterations = %d\n", num_iterations);
      printf("Final Relative Residual Norm = %e\n", final_res_norm);
      printf("\n");
   }

   /* Destroy solver */
   HYPRE_BoomerAMGDestroy(solver);
}
/* PCG */
else if (solver_id == 50)
{
   int num_iterations;
   double final_res_norm;

   /* Create solver */
```

```
   HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

   /* Set some parameters (See Reference Manual for more parameters) */
   HYPRE_PCGSetMaxIter(solver, 1000); /* max iterations */
   HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance */
   HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm as the stopping crite
   HYPRE_PCGSetPrintLevel(solver, 2); /* prints out the iteration info */
   HYPRE_PCGSetLogging(solver, 1); /* needed to get run info later */

   /* Now setup and solve! */
   HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x);
   HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x);

   /* Run info - needed logging turned on */
   HYPRE_PCGGetNumIterations(solver, &num_iterations);
   HYPRE_PCGGetFinalRelativeResidualNorm(solver, &final_res_norm);
   if (myid == 0)
   {
      printf("\n");
      printf("Iterations = %d\n", num_iterations);
      printf("Final Relative Residual Norm = %e\n", final_res_norm);
      printf("\n");
   }

   /* Destroy solver */
   HYPRE_ParCSRPCGDestroy(solver);
}
/* PCG with AMG preconditioner */
else if (solver_id == 1)
{
   int num_iterations;
   double final_res_norm;

   /* Create solver */
   HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

   /* Set some parameters (See Reference Manual for more parameters) */
   HYPRE_PCGSetMaxIter(solver, 1000); /* max iterations */
   HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance */
   HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm as the stopping crite
   HYPRE_PCGSetPrintLevel(solver, 2); /* print solve info */
   HYPRE_PCGSetLogging(solver, 1); /* needed to get run info later */

   /* Now set up the AMG preconditioner and specify any parameters */
   HYPRE_BoomerAMGCreate(&precond);
   HYPRE_BoomerAMGSetPrintLevel(precond, 1); /* print amg solution info */
   HYPRE_BoomerAMGSetCoarsenType(precond, 6);
   HYPRE_BoomerAMGSetOldDefault(precond);
   HYPRE_BoomerAMGSetRelaxType(precond, 6); /* Sym G.S./Jacobi hybrid */
   HYPRE_BoomerAMGSetNumSweeps(precond, 1);
```

```
      HYPRE_BoomerAMGSetTol(precond, 0.0); /* conv. tolerance zero */
      HYPRE_BoomerAMGSetMaxIter(precond, 1); /* do only one iteration! */

      /* Set the PCG preconditioner */
      HYPRE_PCGSetPrecond(solver, (HYPRE_PtrToSolverFcn) HYPRE_BoomerAMGSolve,
                          (HYPRE_PtrToSolverFcn) HYPRE_BoomerAMGSetup, precond)

      /* Now setup and solve! */
      HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x);
      HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x);

      /* Run info - needed logging turned on */
      HYPRE_PCGGetNumIterations(solver, &num_iterations);
      HYPRE_PCGGetFinalRelativeResidualNorm(solver, &final_res_norm);
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n", final_res_norm);
         printf("\n");
      }

      /* Destroy solver and preconditioner */
      HYPRE_ParCSRPCGDestroy(solver);
      HYPRE_BoomerAMGDestroy(precond);
   }
   /* PCG with Parasails Preconditioner */
   else if (solver_id == 8)
   {
      int    num_iterations;
      double final_res_norm;

      int      sai_max_levels = 1;
      double   sai_threshold = 0.1;
      double   sai_filter = 0.05;
      int      sai_sym = 1;

      /* Create solver */
      HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

      /* Set some parameters (See Reference Manual for more parameters) */
      HYPRE_PCGSetMaxIter(solver, 1000); /* max iterations */
      HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance */
      HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm as the stopping crite
      HYPRE_PCGSetPrintLevel(solver, 2); /* print solve info */
      HYPRE_PCGSetLogging(solver, 1); /* needed to get run info later */

      /* Now set up the ParaSails preconditioner and specify any parameters */
      HYPRE_ParaSailsCreate(MPI_COMM_WORLD, &precond);
```

```
      /* Set some parameters (See Reference Manual for more parameters) */
      HYPRE_ParaSailsSetParams(precond, sai_threshold, sai_max_levels);
      HYPRE_ParaSailsSetFilter(precond, sai_filter);
      HYPRE_ParaSailsSetSym(precond, sai_sym);
      HYPRE_ParaSailsSetLogging(precond, 3);

      /* Set the PCG preconditioner */
      HYPRE_PCGSetPrecond(solver, (HYPRE_PtrToSolverFcn) HYPRE_ParaSailsSolve,
                          (HYPRE_PtrToSolverFcn) HYPRE_ParaSailsSetup, precond)

      /* Now setup and solve! */
      HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x);
      HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x);


      /* Run info - needed logging turned on */
      HYPRE_PCGGetNumIterations(solver, &num_iterations);
      HYPRE_PCGGetFinalRelativeResidualNorm(solver, &final_res_norm);
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n", final_res_norm);
         printf("\n");
      }

      /* Destory solver and preconditioner */
      HYPRE_ParCSRPCGDestroy(solver);
      HYPRE_ParaSailsDestroy(precond);
   }
   /* Flexible GMRES with  AMG Preconditioner */
   else if (solver_id == 61)
   {
      int    num_iterations;
      double final_res_norm;
      int    restart = 30;
      int    modify = 1;


      /* Create solver */
      HYPRE_ParCSRFlexGMRESCreate(MPI_COMM_WORLD, &solver);

      /* Set some parameters (See Reference Manual for more parameters) */
      HYPRE_FlexGMRESSetKDim(solver, restart);
      HYPRE_FlexGMRESSetMaxIter(solver, 1000); /* max iterations */
      HYPRE_FlexGMRESSetTol(solver, 1e-7); /* conv. tolerance */
      HYPRE_FlexGMRESSetPrintLevel(solver, 2); /* print solve info */
      HYPRE_FlexGMRESSetLogging(solver, 1); /* needed to get run info later */
```

```
      /* Now set up the AMG preconditioner and specify any parameters */
      HYPRE_BoomerAMGCreate(&precond);
      HYPRE_BoomerAMGSetPrintLevel(precond, 1); /* print amg solution info */
      HYPRE_BoomerAMGSetCoarsenType(precond, 6);
      HYPRE_BoomerAMGSetOldDefault(precond);
      HYPRE_BoomerAMGSetRelaxType(precond, 6); /* Sym G.S./Jacobi hybrid */
      HYPRE_BoomerAMGSetNumSweeps(precond, 1);
      HYPRE_BoomerAMGSetTol(precond, 0.0); /* conv. tolerance zero */
      HYPRE_BoomerAMGSetMaxIter(precond, 1); /* do only one iteration! */

      /* Set the FlexGMRES preconditioner */
      HYPRE_FlexGMRESSetPrecond(solver, (HYPRE_PtrToSolverFcn) HYPRE_BoomerAMG
                           (HYPRE_PtrToSolverFcn) HYPRE_BoomerAMGSetup, precond)


      if (modify)
      /* this is an optional call  - if you don't call it, hypre_FlexGMRESModi
         is used - which does nothing.  Otherwise, you can define your own, sim
         the one used here */
         HYPRE_FlexGMRESSetModifyPC( solver,
                                     (HYPRE_PtrToModifyPCFcn) hypre_FlexGMRESMo


      /* Now setup and solve! */
      HYPRE_ParCSRFlexGMRESSetup(solver, parcsr_A, par_b, par_x);
      HYPRE_ParCSRFlexGMRESSolve(solver, parcsr_A, par_b, par_x);

      /* Run info - needed logging turned on */
      HYPRE_FlexGMRESGetNumIterations(solver, &num_iterations);
      HYPRE_FlexGMRESGetFinalRelativeResidualNorm(solver, &final_res_norm);
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n", final_res_norm);
         printf("\n");
      }

      /* Destory solver and preconditioner */
      HYPRE_ParCSRFlexGMRESDestroy(solver);
      HYPRE_BoomerAMGDestroy(precond);

   }
   else
   {
      if (myid ==0) printf("Invalid solver id specified.\n");
   }

   /* Save the solution for GLVis visualization, see vis/glvis-ex5.sh */
   if (vis)
```

```
   {
      FILE *file;
      char filename[255];

      int nvalues = local_size;
      int *rows = (int*) calloc(nvalues, sizeof(int));
      double *values = (double*) calloc(nvalues, sizeof(double));

      for (i = 0; i < nvalues; i++)
         rows[i] = ilower + i;

      /* get the local solution */
      HYPRE_IJVectorGetValues(x, nvalues, rows, values);

      sprintf(filename, "%s.%06d", "vis/ex5.sol", myid);
      if ((file = fopen(filename, "w")) == NULL)
      {
         printf("Error: can't open output file %s\n", filename);
         MPI_Finalize();
         exit(1);
      }

      /* save solution */
      for (i = 0; i < nvalues; i++)
         fprintf(file, "%.14e\n", values[i]);

      fflush(file);
      fclose(file);

      free(rows);
      free(values);

      /* save global finite element mesh */
      if (myid == 0)
         GLVis_PrintGlobalSquareMesh("vis/ex5.mesh", n-1);
   }

   /* Clean up */
   HYPRE_IJMatrixDestroy(A);
   HYPRE_IJVectorDestroy(b);
   HYPRE_IJVectorDestroy(x);

   /* Finalize MPI*/
   MPI_Finalize();

   return(0);
}

/*--------------------------------------------------------------------------
   hypre_FlexGMRESModifyPCAMGExample -
```

```
    This is an example (not recommended)
   of how we can modify things about AMG that
   affect the solve phase based on how FlexGMRES is doing... For
   another preconditioner it may make sense to modify the tolerance..

 *--------------------------------------------------------------------------*/

int hypre_FlexGMRESModifyPCAMGExample(void *precond_data, int iterations,
                                      double rel_residual_norm)
{


   if (rel_residual_norm > .1)
   {
            HYPRE_BoomerAMGSetNumSweeps((HYPRE_Solver)precond_data, 10);
   }
   else
   {
      HYPRE_BoomerAMGSetNumSweeps((HYPRE_Solver)precond_data, 1);
   }


   return 0;
}
```