TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

# PARALELNÍ VIZUALIZACE V PROSTŘEDÍ PARAVIEW

## Bakalářská práce

*Studijní program:* B2646 – Informační technologie
*Studijní obor:* 1802R007 – Informační technologie

*Autor práce:* **Filip Kulka**
*Vedoucí práce:* Ing. Petr Šidlof, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

# PARALLEL VISUALIZATIONS IN PARAVIEW

## Bachelor thesis

*Study programme:* B2646 – Information Technology
*Study branch:* 1802R007 – Information Technology

*Author:* **Filip Kulka**
*Supervisor:* Ing. Petr Šidlof, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
**Fakulta mechatroniky, informatiky a mezioborových studií**
Akademický rok: **2014/2015**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Filip Kulka** |
| Osobní číslo: | **M12000152** |
| Studijní program: | **B2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Paralelní vizualizace v prostředí ParaView** |
| Zadávající katedra: | **Ústav nových technologií a aplikované informatiky** |

Zásady pro vypracování:

1. Seznamte se s problematikou vizualizace velkých dat a vyzkoušejte si jednoduché vizualizace pomocí softwaru Paraview. Popište architekturu programu v paralelním módu, mechanismus výměny dat a komunikace mezi jednotlivými procesy.
2. Nastudujte a vyzkoušejte si základy práce v operačním systému Linux.
3. Nastudujte a popište architekturu výpočetního clusteru NTI FM Hydra, případně superpočítače Altix UV 100 CIV ČVUT fox.
4. Zkompilujte na clusteru Hydra (případně superpočítači fox) Paraview připravený pro paralelní vizualizace (pvserver + client).
5. Pomocí MPI paralelizace proveďte paralelní vizualizaci velkých dat, distribuovaných na jednotlivých uzlech.
6. Vyhodnoťte zrychlení a snížení paměťových nároků na jeden výpočetní uzel oproti sériovému běhu a analyzujte úzká místa.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 60 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:


**[1] http://www.vtk.org/Wiki/Setting_up_a_ParaView_Server**
**[2] http://www.sandia.gov/ kmorel/documents/sc05/**
**[3] https://computing.llnl.gov/vis/screensteps/Running_ParaView_in_Client-Server_mode.html**
**[4] http://www.itk.org/Wiki/ParaView/Users_Guide/Starting_Parallel_Servers**
**[5] http://elib.uni-stuttgart.de/opus/volltexte/2014/8952/pdf/print.pdf (str. 23-24)**
**[6] http://www.paraview.org/Wiki/ParaView/Users_Guide/Parallel_Rendering**
**[7] https://sites.google.com/site/jordimuela/openfoam/how-to-remote-visualization-using-paraview-in-parallel**
**[8] https://www.scorec.rpi.edu/wiki/Running_ParaView_In_Parallel**


Vedoucí bakalářské práce: **Ing. Petr Šidlof, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **20. října 2014**

Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

# Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2015

Podpis: 

# Poděkování

Mé poděkování patří Ing. Petru Šidlofovi, Ph.D, za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Děkuji také Ing. Jiřímu Hnídkovi, Ph.D, za pomoc při práci se školním clusterem Hydra.

# Abstrakt

Tato bakalářská práce je zaměřená na problematiku paralelních vizualizací pomocí programu ParaView. Lze zde nalézt informace o daném softwaru, potřebném pro paralelní vizualizace na výpočetních clusterech, i instrukce k jeho správnému nainstalování a nastavení. Dále je zde popsána architektura programu ParaView, jeho režimy pro paralelní výpočty a jeho způsob práce s daty. Následně jsou zde zobrazeny výsledky měření časové a paměťové náročnosti při standalone módu a client-server módu v sériové a paralelní konfiguraci. U módu standalone bylo zjištěno několikanásobně větší využití paměti, než při sériovém a paralelním client-server provedení. Při porovnání sériové a paralelní client-server vizualizace bylo shledáno třívteřinové časové zrychlení ve prospěch paralelní vizualizace. Využití paměti bylo v obou případech stejné.

**Klíčová slova:** ParaView, paralelní, vizualizace, OpenMPI

# Abstract

This bachelor thesis is focused on an issue of a parallel visualization in ParaView. The thesis also provides information about software required for parallel visualization on computing clusters, as well as instructions for its proper installation and setup. There is also described an architecture of ParaView, its modes for parallel computing and its way of working with the data. Then there are shown the results of time and memory usage in standalone mode and client-server mode for serial and parallel configuration. In standalone mode, there has been found several times greater memory usage than in serial and parallel client-server configuration. When comparing serial and parallel client-server visualization, it was found that the parallel visualization was 3 seconds faster. Memory usage was same in both cases.

**Keywords:** ParaView, parallel, visualization, OpenMPI

# Obsah

# Seznam symbolů, zkratek a termínů

- **SSH** neboli Secure Shell umožňuje bezpečnou komunikaci mezi dvěma počítači, která se využívá pro zprostředkování přístupu k příkazovému řádku, kopírování souborů a též jakýkoliv obecný přenos dat (s využitím síťového tunelování)
- **GUI** neboli Graphical User Interface je uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.
- **RPM** neboli Red Hat Package Manager je balíčkovací systém pro Linux původně vyvinutý firmou Red Hat.
- **EPEL** neboli Extra Packages for Enterprise Linux je dobrovolná komunita, která vytváří, udržuje a spravuje sadu doplňkových balíčků pro Enterprise Linux, CentOS a Scientific Linux.
- **VTK** neboli Visualization Toolkit je opensourcový software pro 3D počítačovou grafiku, zpracování obrazu a vizualizace.

.

# Seznam obrázků

# Seznam tabulek

# Úvod

Při rozsáhlých paralelních výpočtech se často generují značně obsáhlé soubory s výstupními daty, při nestacionárních výpočtech tak objem dat uložených na disk šplhá na desítky GB. Vizualizace těchto dat je potom značně výpočetně a časově náročná. Při sériovém provedení takto náročných vizualizací dochází kvůli nedostačující systémové paměti klienta velmi často k pádu programu. Z tohoto důvodu je potřeba provádět vizualizace na výkonějších přístrojích. Konkrétně na výpočetním clusteru, který provádí renderování dat místo klienta a tím zmenšuje jeho vytížení paměti. K tomu byl použit jeden z dostupných programů pro postprocessing a vizualizaci vědeckotechnických výpočtů, s názvem ParaView. Dalšími dostupnými programy jsou například open-sourcový VisIt, který je velmi podobný ParaView a komerční EnSight.

Cílem práce bylo seznámit se s problematikou vizualizace velkých dat a vyzkoušet si jednoduché vizualizace pomocí softwaru ParaView. Pomocí MPI paralelizace provést paralelní vizualizaci velkých dat, distribuovaných na jednotlivých uzlech výpočetního clusteru Hydra a následně vyhodnotit zrychlení a snížení paměťových nároků na jeden výpočetní uzel při paralelní vizualizaci oproti sériovému běhu.

# 1 Hydra

Hydra je výpočetní cluster fakulty Mechatroniky, Technické univerzity v Liberci. Slouží pro výuku paralelního programování a testování paralelních aplikací [1].

K Hydře je možné se připojit pouze pomocí SSH na adrese hydra.kai.tul.cz. K připojení je potřeba vlastní účet na výpočetním clusteru, který mi vytvořil jeden ze správců Hydry, Ing. Jiří Hnídek, Ph.D. Z operačního systému Windows jsem byl nucen se připojovat pomocí klienta PuTTY, pro nahrávání souborů na cluster jsem použil program WinSCP.

## *1.1 Technické informace*

Výpočetní cluster Hydra se skládá z 12 uzlů Dell PowerEdge 1950 a 17 uzlů Sun Fire V20z s těmito technickými parametry:

- 12 uzlů Dell PowerEdge 1950 (též frontend)
  - 2x Intel Xeon 5140 2.33GHz/4MB 1333FSB (2 jádra)
  - 4GB RAM 667MHz (4x1HB)
  - 80GB SATA2, 7200 ot. /min, hot plug
  - 2x NIC 1 Gbps, Broadcom NetXtreme II 5708 Gigabit Ethernet NIC
  - DVD ROM
  - Celkem: 24 CPU, 48 jader, 48GB RAM, 960GB HDD
- 17 uzlů Sun Fire V20z (1U)
  - 2x AMD Opteron 252, 2600 MHz (1 jádro)
  - 4 GB RAM
  - 73 GB HDD, 10025 ot. /min, Fujitsu MAT3073NC
  - 1x Dual Ultra320 SCSI, LSI Logic 53c1030 PCI-X
  - 2x NIC 1 Gbps, Broadcom BCM5704
  - DVD-ROM, FDD
  - Celkem: 34 CPU (34 jader), 68 GB RAM, 1.2 TB HDD
- Celková výpočetní kapacita clusteru je 78 jader (46 CPU)
- Rocks Clusters 5.3 - Rolled Tacos
- Linux CentOS 5.4
- propojovací síť 1 Gbps
- přímá 1 Gbps konektivita do Liane, resp. Cesnetu [2]

# 2 Paralelní programování

Paralelní programování je označení konceptu, který umožňuje naprogramovat úlohy schopny současného běhu. Paralelní programovací jazyky, knihovny a API byly vytvořeny pro programování paralelních počítačů. Tyto počítače mohou být rozděleny do několika kategorií, na základě toho jaký používají typ paměti: sdílenou paměť, nebo distribuovanou paměť. Programy využívající sdílenou paměť (OpenMP) komunikují pomocí manipulace s proměnnými ve sdílené paměti. Programy využívající distribuovanou paměť (MPI) používají metodu zasílání zpráv [3].

## *2.1 MPI*

MPI, neboli Message Passing Interface, je knihovna implementující stejnojmennou specifikaci pro podporu paralelních výpočtů na vysoce výkonných clusterech. Jedná se o standardizovaný systém navržený skupinou výzkumných pracovníků z akademické sféry a průmyslu, fungující na široké škále paralelních počítačů. MPI je jazykově nezávislý komunikační protokol. Nejčastěji se však setkáme s implementací v C, C++, Javě, Pythonu a Fortranu. Obojí, jak point-to-point, tak kolektivní komunikace jsou podporovány. Cíle MPI jsou vysoký výkon, škálovatelnost a přenositelnost. MPI zůstává i v dnešní době dominantním modelem používaným při vysoce výkonných paralelních výpočtech.

Ačkoli je standard MPI v oblasti paralelního počítání velmi rozšířený, není jediným. Jeho hlavním „konkurentem“ je OpenMP [4].

### 2.1.1 Point-to-point komunikace

Velká část funkcí v MPI je určena pro realizaci komunikace mezi dvěma procesy. Klasickým příkladem je funkce MPI_Send, která pošle zprávu od jednoho procesu jinému procesu. Často jsou tyto funkce používány na programových architekturách typu master-slave („pán a sluha“), kde je řídící uzel zodpovědný za činnost svých „sluhů“. Typicky master pošle dávky instrukcí nebo dat každému podřízenému, počká až přijdou všechny odpovědi a pak je poskládá do jednoho celku.

### 2.1.2 Kolektivní komunikace

Kolektivní komunikace spočívá v komunikaci mezi všemi procesy v procesové skupině (což může znamenat celý proces, nebo programem definovanou podmnožinu).

Typická funkce kolektivní komunikace je MPI_Bcast (zkratka pro broadcast). Tato funkce vezme data z jednoho uzlu a odešle je všem procesům v procesové skupině. Opačnou operaci provádí funkce MPI_Reduce, která shromáždí všechna data od procesů ze skupiny, provede s nimi předem dané operace (jako například sčítání) a následně uloží výsledek na jednom uzlu [5].

## 2.2 Open MPI

Open MPI je knihovna MPI, která spojuje technologie a zdroje z několika ostatních projektů. Je používaná několika TOP500 superpočítači, včetně Roadrunneru, který byl nejrychlejším počítačem na světě od června 2008 do listopadu 2009 a K computeru, nejrychlejším superpočítačem od června 2011 do června 2012.

Open MPI zastupuje splynutí mezi třemi dobře známými MPI implementacemi:

- FT-MPI (University of Tennessee)
- LA-MPI (Alamos National Laboratory)
- LAM/MPI (Indiana University)

ke kterým přispěl PACX-MPI tým ze Stuttgartské univerzity. Tyto čtyři instituce zahrnovaly zakládající členy vývojového týmu Open MPI.

Poslední stabilní verzí Open MPI je 1.8.4 vydaná 19. prosince 2014 [6].

## 2.3 MPICH

MPICH je původní implementací standardu MPI 1.x. MPICH je volně k dispozici pro Unixové operační systémy (včetně Linuxu a Mac OS X) a Microsoft Windows.

Tato implementace vznikla v Argonne National Laboratory a na vývoji se také podílela Mississippi State University. CH v názvu MPICH bylo odvozeno od "Chameleona", což byla přenosná paralelní programovací knihovna, navržená Williamem Groppem, jedním ze zakladatelům MPICHu.

MPICH je jedna z nejpopulárnějších implementací MPI. Používána jako základ pro vývoj dalších implementací, zahrnující IBM MPI, Intel MPI, Cray MPI, Microsoft MPI, Myricom MPI, OSUMVAPICH/MVAPICH2 a mnoho dalších [7].

## 2.4 OpenMP

OpenMP, neboli Open Multi-Processing, je standard pro programování počítačů se sdílenou pamětí. OpenMP usnadňuje vytváření vícevlákenných programů v programovacích jazycích Fortran, C a C++.

Jedná se o soustavu direktiv s implmentací multithreadingu. Multithreading je způsob paralelizace, ve které hlavní vlákno (master) vytváří podle potřeby skupinu podvláken (sluhy), mezi které systém rozděluje jednotlivé úlohy. Ty poté běží souběžně s běhovým prostředím, které vlákna alokuje různým procesorům.

Aktuální stabilní verze OpenMP je 4.0, vydána 23. července 2013 [8].

# 3 ParaView

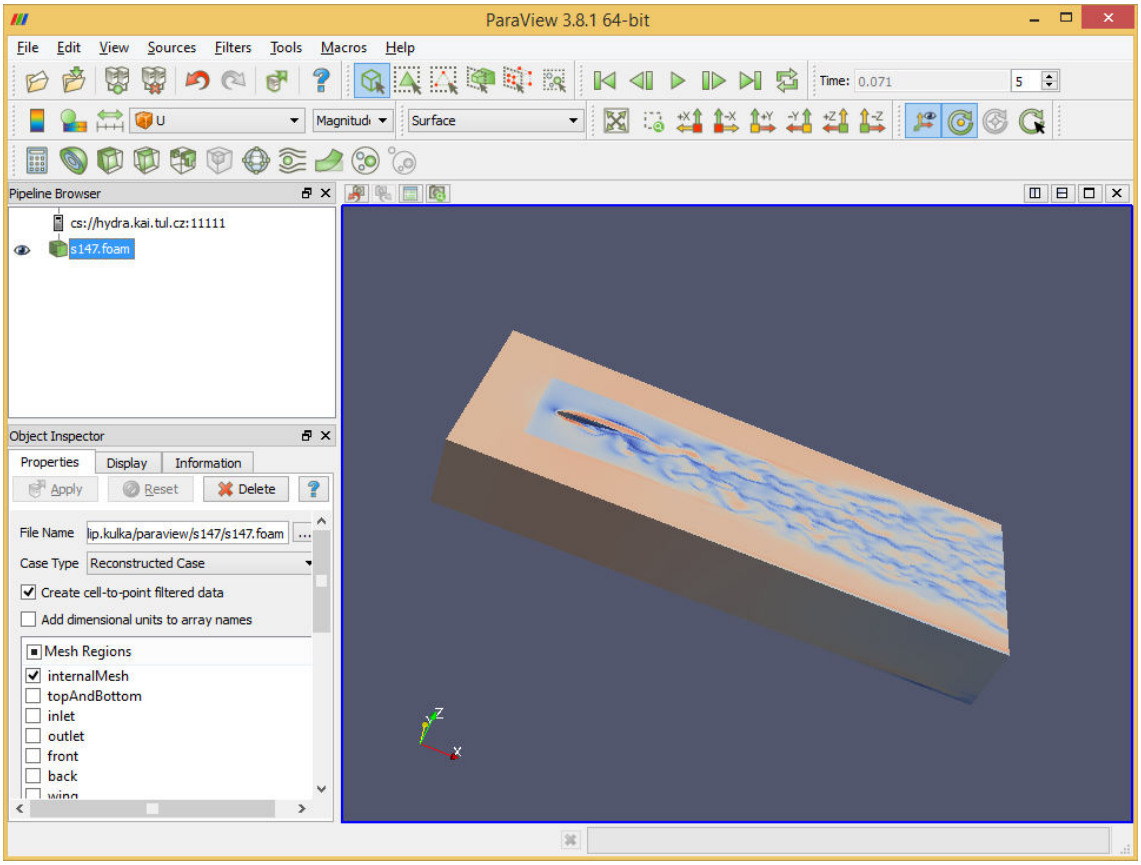
ParaView je volně dostupný software určený pro modelování dat ve dvou a hlavně třech prostorových dimenzích, včetně animování v čase. ParaView byl vyvinut pro analýzu extrémně velkých datových souborů pomocí distribuované paměti výpočetních strojů. Lze spustit na superpočítačích a to k analýze datových souborů ohromných velikostí, ale také na noteboocích pro výpočet menších dat. Je založen na klient-server architektuře, která usnadňuje vzdálenou vizualizaci datových souborů. Stojí na vizualizační knihovně VTK (Visualization Tool Kit). ParaView je aplikace určená pro datový paralelismus na sdílené paměti nebo distribuované paměti multipočítačů a clusterů. Jedná se o multiplatformovou aplikaci, je tedy možné ji používat jak na platformě Windows, Mac OS X, tak i Linux, kde může s podporou knihovny MPI vizualizovat rozměrná data paralelizovaně.

Mezi hlavní cíle vývojového týmu ParaView patří následující:

- Rozvíjet opensourcovou, multiplatformní vizualizační aplikaci
- Podporovat distribuované výpočetní modely pro zpracování rozsáhlých datových souborů
- Vytvořit flexibilní a intuitivní uživatelské rozhraní
- Rozvíjet rozšiřitelnou architekuru programu založenou na open standard standardech

Projekt se jménem ParaView vznikl v roce 2000 za spolupráce Los Alamos National Laboratory a společností Kitware. První verze byla vydána v říjnu roku 2002 pod označením ParaView 0.6. Aktuální verze dostupná ke stažení je verze 4.3.1.

Nezávisle na ParaView začala v prosinci roku 2001 společnost Kitware vyvíjet internetový vizualizační systém. Tento projekt byl financován výzkumnou laboratoří Spojených států amerických a nakonec se z něj stal ParaView Enterprise Edition. A právě ParaView Enterprise Edition dále významně přispěl k rozvoji klient-serverové architektuře samotného ParaView [9, 10].

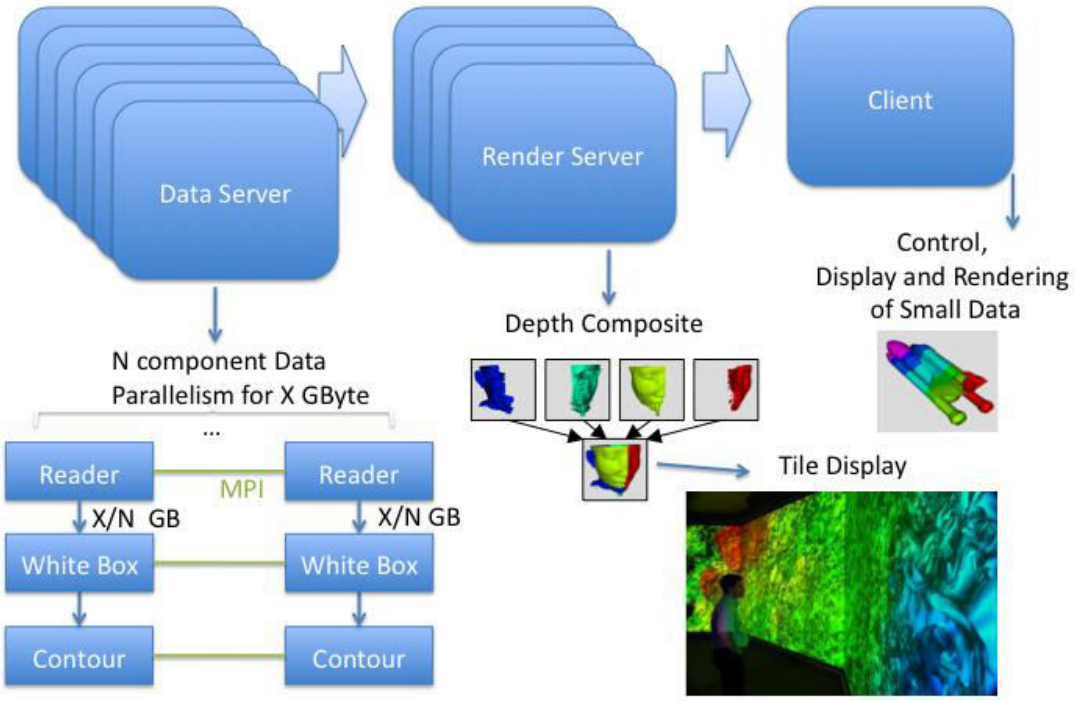



**Obrázek 1: Architektura programu ParaView s daty s147**

## 3.1 Architektura programu ParaView

ParaView je navržen tak, aby dobře pracoval v režimu klient/server. Tato výhoda se nejvíce projeví při práci se vzdáleným vysoce výkonným clusterem. ParaView má tři hlavní logické komponenty: klienta, datový server, a vykreslovací (render) server (viz Obrázek 2). Klient je odpovědný za GUI, jedná se o rozhraní mezi uživatelem a ParaView jako celkem. Datový server čte soubory a zpracovává data přes tzv. pipeline. Vykreslovací server má zpracovaná data a vykresluje je do podoby, která je zobrazena uživateli.

Tyto tři logické komponenty nemusí být fyzicky odděleny. Logické části mohou být vloženy ve stejné aplikaci, což odstraňuje komplikace s jejich vzájemnou komunikací.




**Obrázek 2: Architektura paralelního ParaView [11]**

### 3.1.1 Klient

Klient je zodpovědný za uživatelské rozhraní aplikace. Kontroluje vytváření objektu, provedení a smazání na serverech, ale neobsahuje žádná data. Komponenta klient je vždy sériový program, který řídí součásti serveru prostřednictvím Server Manager API.

### 3.1.2 Datový server

Datový server je primárně postaven z VTK knihoven, zdrojů a filtrů. Je zodpovědný za čtení a generování dat, jejich zpracování a produkci geometrických modelů, které následně zobrazuje vykreslovací server. Datový server využívá datového paralelismu k rozdělení dat přidáním tzv. ghost levels kolem oddílů dle potřeby (popsáno v kapitole 4.1 Ghost levels). Každý proces datového serveru má identickou VTK pipeline a každému procesu je řečeno, kterou část dat má načíst a zpracovat. Díky rozdělení dat je ParaView schopen využít celou systémovou paměť a tím umožnit zpracování velkých dat.

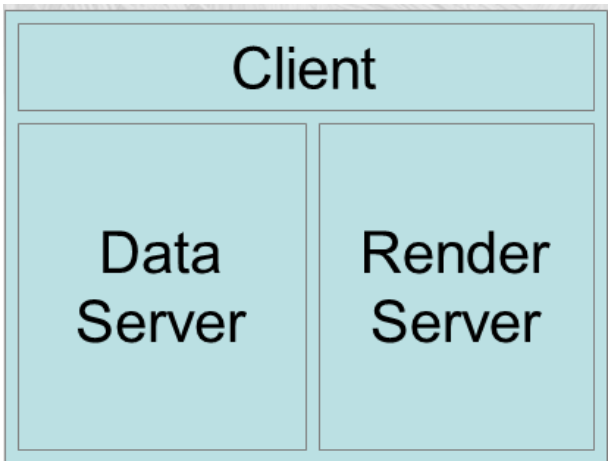
### 3.1.3 Vykreslovací server

Vykreslovací server je zodpovědný za vykreslování geometrie. Stejně jako datový server může být vykreslovací server spuštěn paralelně a má stejné vizualizační pipeline ve všech svých procesech. S možností spuštění odděleně od datového serveru přichází možnost optimální dělby práce mezi výpočetními uzly. Většina velkých výpočetních clusterů je primárně využívána pro spouštění velkých simulací a nemají hardwarové vykreslovací zdroje. Vzhledem k tomu, že není potřeba přenášet velké datové soubory na oddělený vizualizační systém, datový server může být spouštěn na stejném clusteru, na kterém byla původní simulace. Vykreslovací server může být spuštěn na samostatném vizualizačním clusteru, který obsahuje hardwarové vykreslovací zdroje [12].

## *3.2 Způsoby spuštění ParaView*

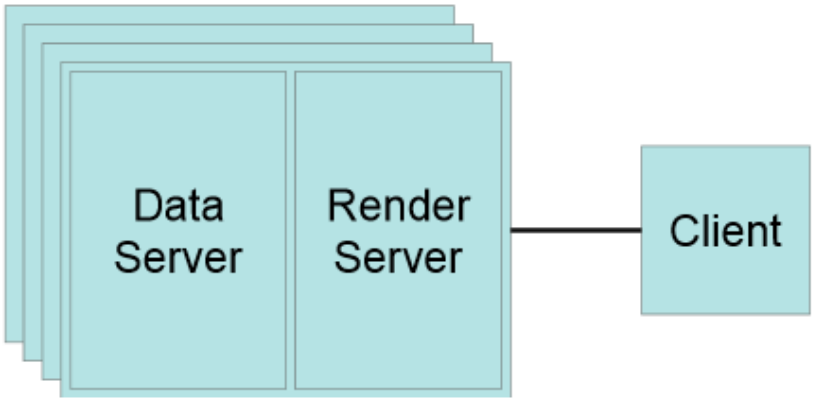
Existují tři způsoby, ve kterých lze ParaView spustit.

První způsob se nazývá *standalone mód*. V tomto módu jsou klient, datový server a renderovací server spojeni do jedné aplikace (viz Obrázek 3). Po spuštění ParaView se klient připojí k tzv. „built-in serveru“.



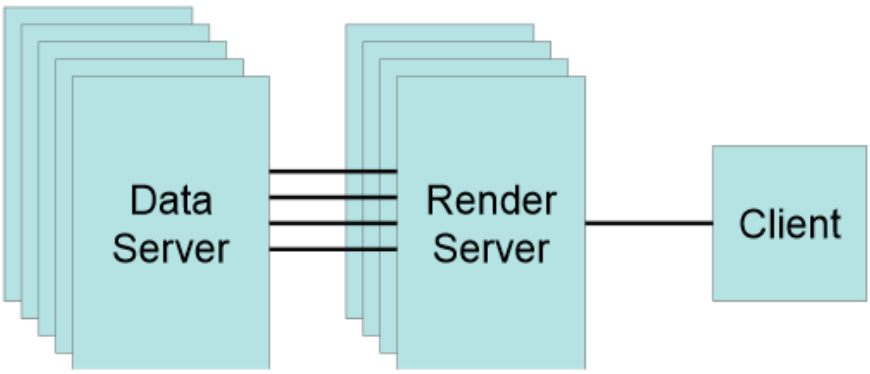

**Obrázek 3: Režim standalone [13]**

Druhá možnost se nazývá *client-server mód*. V tomto režimu uživatel spustí pvserver na paralelním přístroji a připojí se k němu s klientem (viz Obrázek 4). Program pvserver v sobě obsahuje jak datový server, tak renderovací server, tudíž zde probíhá zpracování dat i jejich následné vykreslování. Klient a server jsou mezi sebou připojeni pomocí soketu, u kterého je předpokládaný pomalý přenos při komunikaci, proto je zde omezen přenos dat na minimum.




**Obrázek 4: Režim client-server [13]**

Třetí mód se nazývá *client-render server-data server*. V tomoto režimu jsou všechny tři komponenty jako oddělené programy (viz Obrázek 5). Stejně jako v client-server režimu, i zde je klient připojen k renderovacímu serveru přes soket. Renderovací server a datový server jsou propojeni pomocí několika soketů, jeden pro každý proces renderovacího serveru. Přenos dat těmito sokety je opět minimalizován.



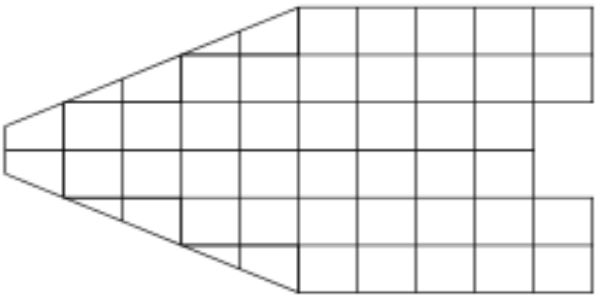

**Obrázek 5: Režim client-render server-data server [13]**

I když je client-render server-data server podporován, je téměř nepoužíván. Hlavní nápad pro vytvoření tohoto režimu bylo využití výhody různorodého prostředí, kde první server je vysoce výpočetně výkonný a druhý obsahuje grafický hardware. Avšak v praxi jsou veškeré tyto výhody téměř znehodnoceny pomalým přenosem dat mezi datovým a renderovacím serverem [13].

Serverové programy jsou datově paralelní programy, které mohou být spuštěny jako soubor nezávislých procesů běžících na různých procesorech. Procesy používají MPI pro koordinaci jejich činnosti, protože každý procesor pracuje na různé části dat.

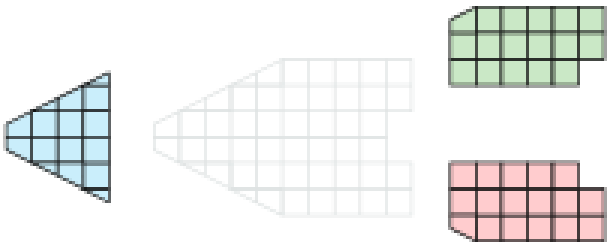
# 4 Algoritmy paralelní vizualizace

Data, se kterými při paralelních vizualizacích pracujeme, jsou obsažena v síti, což znamená, že jsou předem rozdělena na malé elementy (viz Obrázek 6). Můžeme provádět vizualizace na distribuovaných paralelních strojích přiřazením elementů jednotlivým procesům. Tuto práci za nás automaticky obstarává ParaView. Pro ukázku jsem vybral tuto jednoduchou síť.



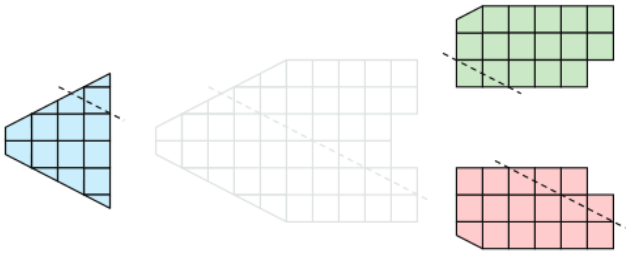
**Obrázek 6: Datová síť [14]**

Nyní chceme na této síti provést vizualizace pomocí třech procesů. Můžeme rozdělit jednotlivé elementy sítě do tří částí, na následujícím obrázku jako modré, zelené a růžové (viz Obrázek 7).



**Obrázek 7: Rozdělní dat do tří částí [14]**

Algoritmy nyní budou pracovat jednoduše tak, že každý proces bude nezávisle vykonávat výpočty na své vlastní části sítě (viz Obrázek 8). Například filtr Clip, pracuje následovně.

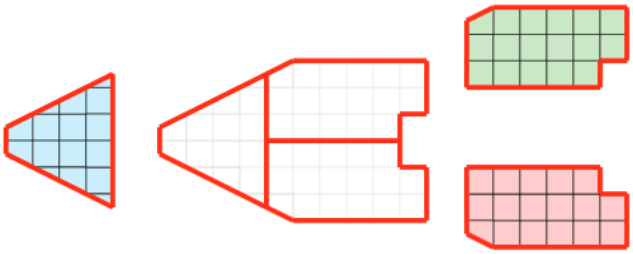


**Obrázek 8: Použití filtru Clip [14]**

Každý proces použije nezávisle na ostatních filtr Clip na svou část sítě. Po dokončení jsou výsledná data stejná, jako kdybychom na ně použili Clip sériově. Pokud složíme jednotlivé elementy zpět k sobě, uvidíme, že operace proběhla úspěšně a elementy jsou na správných místech [15].

## 4.1 Ghost levels

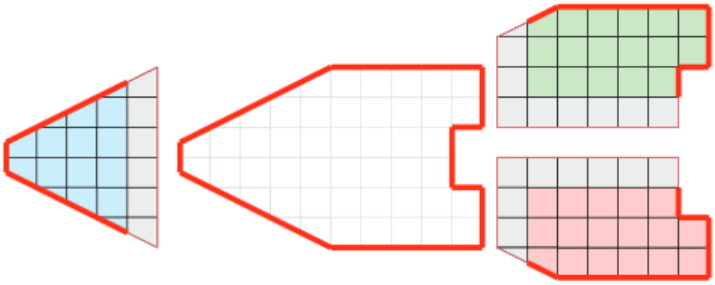
Bohužel některé vizualizační algoritmy spuštěné na rozdělené síti neskončí vždy správným výsledkem. Například algoritmus external faces, který najde všechny elementy patřící do jedné předem rozdělené buňky a pracuje s nimi jako s celkem (viz Obrázek 9).



**Obrázek 9: Použití algoritmu external faces [14]**

Zde nastává problém. Pokud každý proces pracuje nezávisle na ostatních se svou částí dat, mnoho vnitřních částí je nesprávně posouzeno jako vnější části. To se stává, když jeden element má „souseda“ v další části dat jiného procesu. Procesy nemají přístup k datům z jiných částí, proto zde není žádná možnost, aby věděly o tom, že tyto sousední elementy existují.

Řešení ParaView a dalších paralelně vizualizačních systémů je používání takzvaných *ghost cells*. Jedná se o elementy sítě, které jsou přiřazeny jednomu procesu, ale patří jinému. Pro toto řešení musíme nejdříve nalézt všechny sousedící elementy v každé rozdělené části. Následně se tyto elementy nakopírují do sousedící části a označí se jako ghost cell. Na Obrázek 10 jsou zobrazeny šedou barvou.

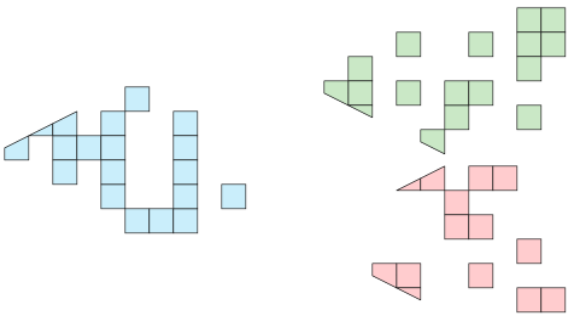


**Obrázek 10: Znázornění ghost cells [14]**

Když spustíme algoritmus external faces s těmito ghost cells, uvidíme, že jsou stále některé interní elementy špatně identifikovány jako externí. Avšak všechny tyto špatně identifikované části jsou označeny jako ghost cells a ParaView je od ostatních dat odřízne. Zůstanou tedy správná data [15].

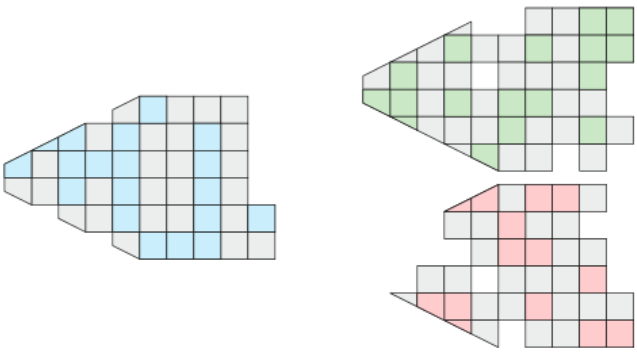
## 4.2 Rozdělení dat

Data zobrazena na předchozích příkladech jsou prostorově souvisle rozdělena. To znamená, že se všechny elementy sítě nachází v jednom kompaktním prostoru. Existují však i další způsoby rozdělení dat, jako například náhodné rozdělení (viz Obrázek 11).



**Obrázek 11: Náhodně rozdělená data [14]**

Tento způsob rozdělení dat má pár dobrých vlastností. Je jednoduché náhodným rozdělením vytvořit strukturu dat a zároveň vyvažuje paměťové zatížení. Na druhou stranu obsahuje vážné problémy s ohledem na již zmiňované ghost cells.



**Obrázek 12: Náhodně rozdělená data v síti [14]**

Na **Error! Reference source not found.** vidíme, že téměř celou síť je potřeba ošetřit pomocí ghost cells algoritmu. Tímto odpadají jakékoliv výhody paralelního zpracování, jelikož samotné zpracování ghost cells je velmi časově náročné. To je hlavním důvodem, proč se náhodné rozdělení dat v ParaView nepoužívá [14].

# 5 Nastavení serveru

Klient programu ParaView je sériová aplikace, kterou lze spustit příkazem paraview. Server je však paralelní MPI program, který musí být spuštěn jako paralelní proces. Různé implementace MPI mohou mít rozdílné způsoby jak spustit paralelní programy, ale nejčastějším způsobem je příkaz mpirun. Na školním serveru Hydra se nachází více verzí MPI, proto je lepší specifikovat plnou cestu:

```
/opt/openmpi/bin/mpirun
```

Server ParaView je téměř vždy možno pustit příkazem pvserver. Tímto se spustí samotný pvserver bez jakéhokoli nastavení – sériově. Opět je však potřeba lépe definovat o jaký pvserver se jedná, v této bakalářské práci se pracovalo s verzí ParaView přeloženou s knihovnou openmpi:

```
/usr/lib64/openmpi/bin/pvserver_openmpi
```

V našem případě se potřebujeme připojit na více uzlů výpočetního serveru Hydra. K tomu slouží následující přepínač „-np 4“, kde -np značí počet procesů (number of processes) a číslo 4 znázorňuje jejich počet. Nyní dokážeme spustit pvserver připravený pro paralelní výpočty. S touto konfigurací by se však všechny výpočty prováděly na frontendu Hydry a ostatní uzly by byly nevyužity. Proto přidáme další přepínač -machinefile, který přímo definuje uzly, na kterých se pvserver spustí. Příkaz vypadá takto:

```
-machinefile /home/filip.kulka/paraview/machines.txt
```

Soubor machines.txt má na každém řádku vypsán jeden uzel, ke kterému se má později ParaView připojit. Dále může obsahovat doplňující informace, jako počet jader procesoru (slots=4). Náš soubor machines.txt má následující strukturu:

```
hydra.kai.tul.cz

compute-0-1

compute-0-2

compute-0-3
```

První uzel hydra.kai.tul.cz je tzv. master a compute-0-1 až compute-0-3 jsou jeho „otroci“ (popsáno v kapitole 2.1.1 Point-to-point komunikace).

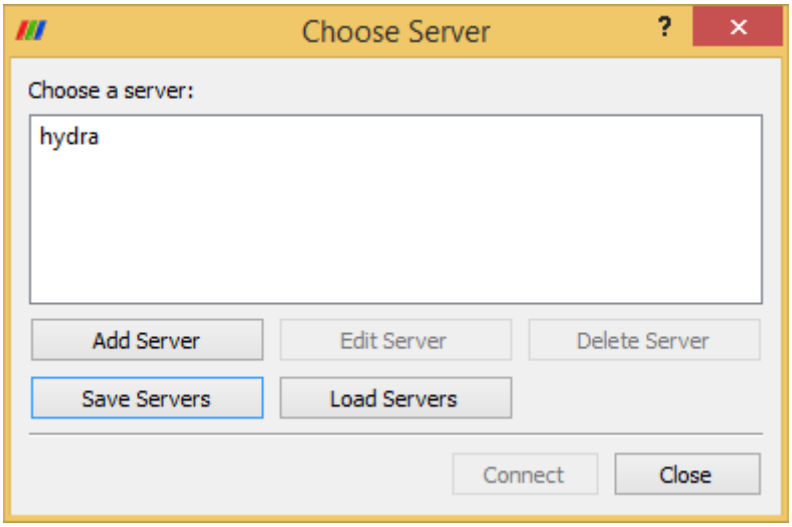
Přidání přepínače „--use-offscreen-rendering“ nám zajistí, že renderování dat obstará master a klient se stará pouze o složení výsledných dat.

Výsledný příkaz pro spuštění pvserveru na 4 nodech vypadá takto:

```
/opt/openmpi/bin/mpirun -np 4 –machinefile
/home/filip.kulka/paraview/machines.txt
/usr/lib64/openmpi/bin/pvserver_openmpi --use-offscreen-
rendering
```

## 5.1 Připojení k serveru

Starší verze ParaView vyžadovaly od uživatele ruční spuštění serveru a klienta. Uživatel byl nucen zadat protokoly připojení a parametry přímo do příkazového řádku. ParaView od verze 3 zlepšuje uživatelský komfort tím, že umožňuje uživatelům spustit klienta a k serveru se připojit pomocí grafického rozhraní klienta.

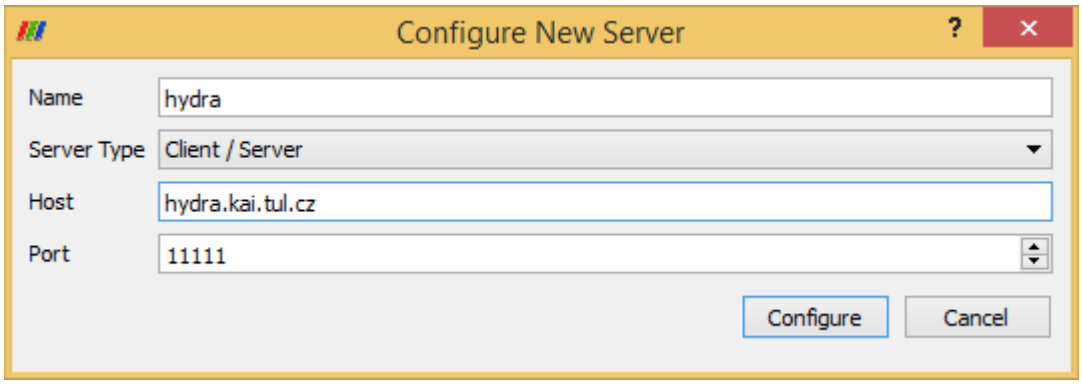



**Obrázek 13: Výběr serveru**

Přidání nového serveru probíhá přes File – Connect a následně Add Server. Pole Name je libovolné. U výběru Server Type jsou následující možnosti:

- Client/Server: Připojí klienta ParaView k serveru
- Client/Server (reverse connection): Připojí server ke klientovi
- Client/Data Server/Render Server: Připojí klineta odděleně k datovému a renderovacímu serveru
- Client/Data Server/Render Server (reverse connection): Připojí oba, datový i renderovací server ke klientovi

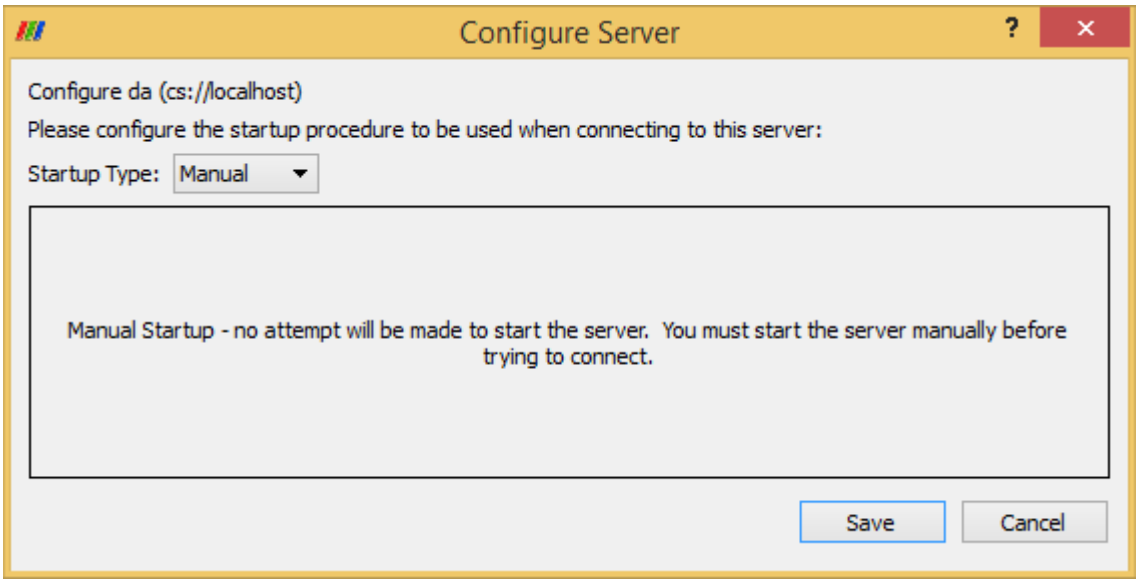
Možnost reverse connection znamená, že se server připojí ke klientovi, místo toho, aby se klient připojil k serveru. To je občas nezbytné, pokud je server za firewallem. Servery jsou většinou spouštěny s více procesy a na jiných přístrojích, než je spuštěn klient.




**Obrázek 14: Konfigurace nového serveru**

V následující kolonce Port bylo ponecháno 11111, což je defaultní hodnota pro datový server. Při připojení na renderovací server, bylo by potřeba tuto hodnotu změnit na 22221.

Po odkliknutí tlačítka Configure je k dispozici nabídka způsobu spuštění serveru. Jedna z možností je automatické spuštění po zapnutí ParaView pomocí příkazu, my jsme vybrali druhou možnost, manuální spuštění.




**Obrázek 15: Ruční spuštění serveru mimo ParaView**

# 6 Kompilace ParaView

Na školním výpočetním clusteru Hydra se nachází operačí systém CentOS verze 5.6. Z důvodu kompatibility knihoven a nastavením jsem nejdříve zkompiloval ParaView na svém počítači pod CentOS stejné verze, vytvořeným pomocí virtualizačního nástroje Virtualbox, abych mohl tyto zkušenosti a již vyřešené problémy použít na Hydře a nezpůsobil na serveru žádnou újmu.

Nejdříve jsem stáhl a zkompiloval OpenMPI verze 1.8.4., která se stará o paralelní spuštění pvserveru. Následně jsem provedl instalaci Pythonu verze 2.7.6:

```
# wget http://python.org/ftp/python/2.7.6/Python-
2.7.6.tar.xz
# tar xf Python-2.7.6.tar.xz
# cd Python-2.7.6
# ./configure --prefix=/usr/local --enable-
unicode=ucs4 --enable-shared LDFLAGS="-Wl,-rpath
/usr/local/lib"
# make
# make install
```

Poté se však zobrazila systémová chyba, že na mém CentOS chybí potřebné balíčky. Po pročtení internetové diskuse se stejným problémem jsem doinstaloval RPM a následující EPEL balíčky:

```
# yum localinstall --nogpgcheck
http://download1.rpmfusion.org/free/el/updates/6/i386/
rpmfusion-free-release-6-1.noarch.rpm
http://download1.rpmfusion.org/nonfree/el/updates/6/i3
86/rpmfusion-nonfree-release-6-1.noarch.rpm

# rpm -Uvh http://mirrors.kernel.org/fedora-
epel/6/i386/epel-release-6-8.noarch.rpm
```

Následně přišla na řadu instalace FFmpeg verze 2.3.3., který potřeboval následující prerekvizity:

```
# yum install gcc-c++ make yasm pkgconfig libXext-
devel libXfixes-devel x264-devel zlib-devel
```

Samotná instalace FFmpeg probíhala pomocí tohoto kódu:

```
# wget http://ffmpeg.org/releases/ffmpeg-
2.3.3.tar.bz2
# tar xvfvj ffmpeg-2.3.3.tar.bz2
# cd ffmpeg-2.3.3
# ./configure --enable-shared --enable-nonfree --
enable-gpl --enable-decoder=aac --enable-libx264 --
enable-demuxer=mov --enable-x11grab --enable-zlib --
enable-protocol=http --enable-filter=aformat --
enable-filter=volume --enable-filter=aresample
# make
# make install
```

Další knihovny, potřebné k instalaci, byly Mesa 3D, konkrétně verze 7.7.1.:

```
# yum install libdrm
# tar zxf MesaLib-7.7.1.tar.gz
# ./config
# make realclean
# make linux-x86-64
```

Následovala instalace qmake a Qt knihoven, které jsou potřebné pro překlad uživatelského rozhraní:

```
# yum install qt-devel
# yum --skip-broken groupinstall 'Development Tools'
# yum install openssl-devel
# yum install libXext-devel
# yum install libXt-devel
# curl -C - -O http://releases.qt-
project.org/qt4/source/qt-everywhere-opensource-src-
4.8.2.tar.gz
# tar -xvf qt-everywhere-opensource-src-4.8.2.tar.gz
# cd qt-everywhere-opensource-src-4.8.2
# configure --prefix=/usr/local/qt -prefix-install -
openssl -confirm-license
# qmake
# qmake install
```

Posledním krokem před kompilací samotného ParaView byl build vlastního Cmaku, konkrétně verze 2.8.8.:

```
# cd $HOME
# wget http://www.cmake.org/files/v2.8/cmake-
2.8.8.tar.gz
# tar xvfz cmake-2.8.8.tar.gz
# cd cmake-2.8.8
# ./configure --prefix=$HOME/software
# make
# make install
```

Nakonec bylo potřeba nainstalovat následující seznam balíčků:

```
# yum libphonon-dev libphonon4 qt4-dev-tools libqt4-
core libqt4-gui qt4-qmake libxt-dev g++ gcc cmake-
curses-gui libqt4-opengl-dev mesa-common-dev openmpi-
common openmpi-bin libopenmpi-dev python-dev
```

Nyní se dostaneme k samotnému překladu programu ParaView, kde jsem postupoval podle návodu [16]. Nejdříve stáhneme vývojovou verzi z gitu:

Vytvoření složky pro stažení:

```
# mkdir $HOME/projects
# cd $HOME/projects
```

Stažení zdrojového kódu:

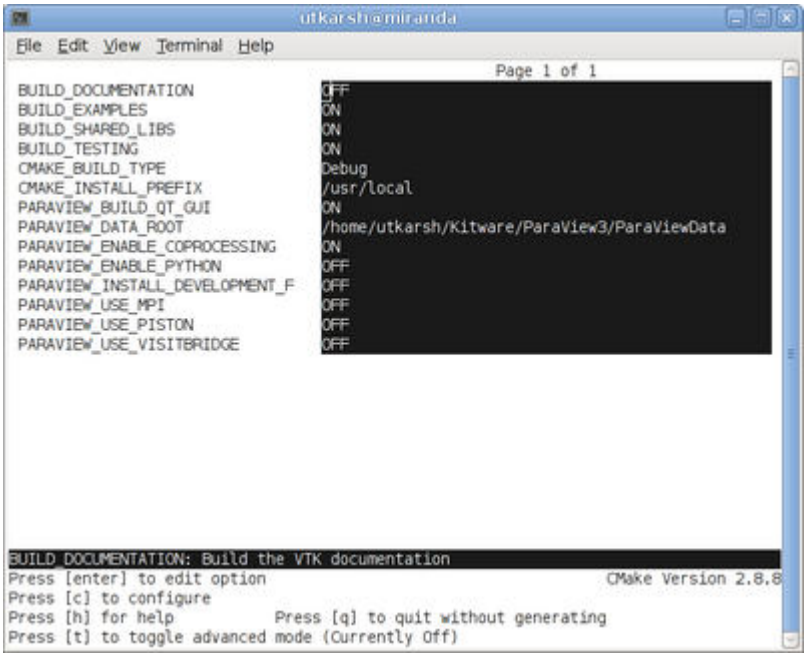
```
# git clone git://paraview.org/ParaView.git ParaView
# cd ParaView
# git checkout -b trunk origin/master
# git submodule init
# git submodule update
```

Update kódu:

```
# git fetch origin
# git rebase origin/master
# git submodule update
```

Nyní přecházíme k samotnému nastavení ParaView pomocí již dříve staženého Cmake. Zde je důležité nepřekládat do zdrojové složky ParaView, ale použít jinou složku (v našem případě ParaView3).

```
# mkdir $HOME/projects/ParaView-bin
# cd $HOME/projects/ParaView-bin
# ccmake $HOME/projects/ParaView3
```




**Obrázek 16: Nastavení při buildu ParaView [16]**

Pokud jsme správně nainstalovali všechny potřebné balíky, Cmake si doplní jejich cesty a defaultní nastavení. Jedinou věc, na kterou při tomto kroku musíme dát pozor, je nastavení MPI, konkrétně nastavit proměnnou prostředí PARAVIEW_USE_MPI na hodnotu ON. Hodnoty nastavujeme pomocí klávesy c (configure). Pokud jsou všechny hodnoty vyplněny a nakonfigurovány, zobrazí se možnost použití klávesy g (generate), kterou vygenerujete konfigurační soubor potřebný k buildu programu. Pokud vše proběhlo v pořádku, stačí jen zadat:

```
make
make install
```

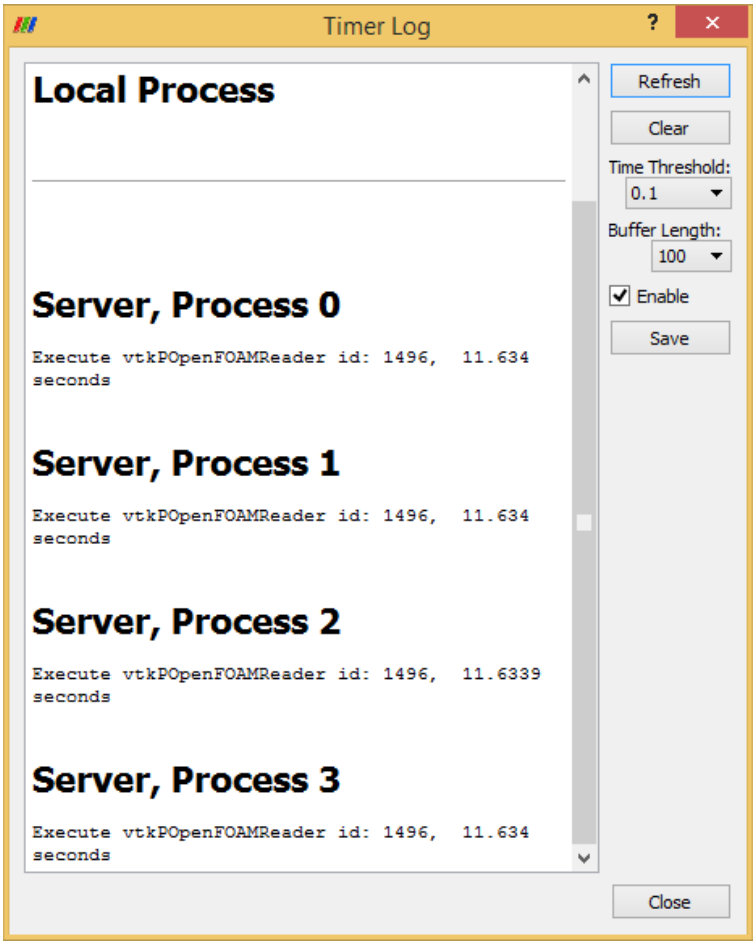
Nyní máme zkompilovaný ParaView s možností paralelní vizualizace. Alternativní postup instalace ParaView s podporou paralelizace je dostupný v dokumentu [17].

# 7 Porovnání časové a paměťové náročnosti

Při vizualizacích na výpočetním serveru Hydra jsem měřil dva faktory:

- vytížení paměti
- časovou náročnost

Pro získání časových hodnot jsem použil rozšíření zabudované přímo v programu ParaView, zvané Timer Log. Tato funkce vypisuje všechny provedené operace a jejich dobu trvání. Na Obrázek 18 můžeme vidět výpis Timer Logu po načtení dat při paralelní vizualizaci na 4 nodech. Operace načtení dat je zobrazena jako vtkPOpenFOAMReader. Vidíme, že probíhala pod všemi čtyřmi procesy a trvala 11.634 sekundy.

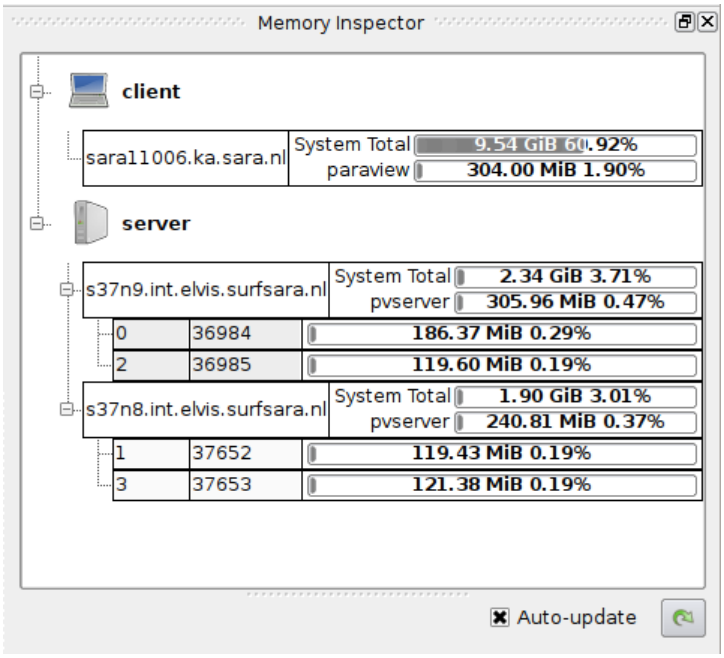



**Obrázek 17: Zobrazení výpisu Timer Log**

Timer Log nám zobrazí i informace o tom, který proces provádí jakou operaci, například lokální proces-klient při načítání dat neprovádí nic.

Pro získání hodnot o využití paměti jsem bohužel nemohl použít další funkci programu ParaView, zvané Memory Inspector. To se nachází až v novějších verzích ParaView (od verze 3.98), které kvůli kompatibilitě s nainstalovanými knihovnami na Hydře nelze použít. Zvolil jsem tedy možnost sledování paměťových nároků pomocí příkazu *top,* který zobrazuje právě probíhající procesy a tím jsem získal využití paměti na daném uzlu.

Na výpisu z funkce Memory Inspector, použitém na nizozemském výpočetním serveru Elvis (Obrázek 18), vidíme využití paměti lokálního stroje a serveru, ke kterému je ParaView připojený. V tomto případě se jedná o paralelní vizualizaci se čtyřmi procesy, konkrétně o připojení na dva nody a u obou z nich na dva procesory.




**Obrázek 18: Výpis z Memory Inspectoru [18]**

Měření vždy probíhala ve třech krocích, kde jsem u každého z nich zaznamenával naměřené hodnoty. Konkrétně se jednalo o načtení dat, použití filtru Clip a nakonec zobrazení dat v režimu Surface with Edges.

K měření jsem použil výpočetní server Hydra (popsán v kapitole 1.1 Technické informace) a notebook Lenovo Z500, jehož parametry jsou následující:

- Procesor: Intel Core i5 3210M (2.5/3.1GHz)
- RAM: 4GB DDR3/1 600 MHz
- Grafická karta: NVIDIA GeForce GT 645M

- Operační systém: Windows 8.1

## 7.1 Použitá data

Nejdříve proběhla měření s menšími daty s147. Můj následný předpoklad po těchto měřeních byl takový, že při použití větších dat by se měly projevit rozdíly mezi paralelními a sériovými výpočty ve prospěch paralelních, s ohledem na časovou náročnost vizualizace. Proto jsem od vedoucího této bakalářské práce, Ing. Petra Šidlofa, Ph.D dostal větší data. Měření tedy probíhala na dvou následujících typech dat:

- s147: se simulací nestlačitelného proudění kolem leteckého profilu. V síti 1,6 miliónu elementů.
- s183b: se simulací proudění v lidském hrtanu. V síti 5,4 miliónu elementů.

## 7.2 Výsledky porovnání

Celé měření bylo rozděleno do dvou částí. V první části bylo porovnáváno využití paměti klienta (Tabulka 1), ve druhé části využití paměti výpočetního serveru (Tabulka 2) a časová náročnost vizualizace (Tabulka 3).

Nejdříve jsem měřil využití paměti počítače při vizualizaci v režimu standalone (popsáno v kapitole 3.2 Způsoby spuštění ParaView), sériové client-server vizualizaci na Hydře a paralelní client-server vizualizaci na dvou nodech clusteru Hydra.

**Tabulka 1: Vytížení paměti klienta**

| Využití paměti | Standalone | client-server (1 proces) | client-server (2 procesy) |
|---|---|---|---|
| **Načtení dat s147** | 644 MB | 106 MB | 103 MB |
| **Načtení dat s183b** | 1,14 GB | 117 MB | 116 MB |

Z naměřených dat vidíme, že využití paměti v těchto třech scénářích je velmi rozdílné. Při vizualizaci v režimu standalone si ParaView načítá data z pevného disku počítače a uchovává si je, proto se dostáváme na využití 644 MB po načtení dat s147 a na využití 1,14 GB po načtení dat s183b. Na druhou stranu, při obou client-server vizualizacích na Hydře, se data nachází na serveru. Tím pádem ParaView nevyužívá

tolik systémové paměti počítače a pracuje pouze s vyrenderovanými daty, které mu pošle server k zobrazení. Samotný klient po spuštění využívá 90 MB operační paměti. V tomto případě pracuje s výsledky o velikosti 16 MB u dat s147 a 27 MB u dat s183b.

Ve druhé části jsem měřil maximální využití paměti serveru při sériovém client-server připojení a paralelním client-server připojení na dva a čtyři uzly.

**Tabulka 2: Využití paměti clusteru s daty s147 a s183b**

| Využití paměti | client-server (1 proces) | client-server (2 procesy) | client-server (4 procesy) |
|---|---|---|---|
| **Načtení dat s147** | 503 MB | 499 MB a 49 MB | 507 MB, 53 MB, 53 MB a 53MB |
| **Načtení dat s183b** | 1,2 GB | 969 MB a 53 MB | 969 MB, 53 MB, 53 MB a 53 MB |
| **Použití filtru Clip na data s147** | 612 MB | 592 MB a 57 MB | 621 MB, 57 MB, 57 MB a 57 MB |
| **Použití filtru Clip na data s183b** | 1,36 GB | 1,37 GB a 57 MB | 1,37 GB, 57 MB, 57 MB a 57 MB |
| **Surface with Edges s daty s147** | 620 MB | 596 MB a 57 MB | 616 MB, 57 MB, 57 MB a 57 MB |
| **Surface with Edges s daty s 183b** | 1,77 GB | 1,77 GB a 69 MB | 1,74 GB, 57 MB, 57 MB a 57 MB |

V tabulce č. 2 je zobrazeno paměťové vytížení Hydry. U obou paralelních provedení reprezentuje první číslo frontend clusteru Hydra. Zbylé hodnoty jsou naměřeny z ostatních nodů, na kterých vizualizace probíhala. Můžeme zde názorně vidět, jak ParaView při paralelní vizualizaci pracuje. Jedná se o master-slave architekturu (popsáno v kapitole 2.1.1 Point-to-point komunikace), kde master rozděluje práci svým sluhům. V tomto případě je využití paměti u sluhů stejné jak při výpočtu, tak při nečinnosti aplikace. Celou dobu si drží, až na odchylku několika MB, svá data.

Master však pracuje jinak. Při nečinnosti aplikace nevyužívá žádná data, pokud však uživatel provede v klientovi ParaView libovolnou operaci potřebující výpočet, „probudí se“ a jeho paměťové nároky rapidně vzrostou.

Z výsledných hodnot v tabulce vidíme, že paměťové nároky master procesu jsou prakticky stejné bez ohledu na to, na kolik uzlů se úloha paralelizuje.

Následně jsem se zaměřil na časovou náročnost při výše zmiňovaných operacích a konfiguracích. Tyto data jsem získával z funkce Timer Log (popsáno v kapitole 7 Porovnání časové a paměťové náročnosti).

**Tabulka 3: Časová náročnost vizualizace s daty s147 a s183b**

| Časová náročnost | client-server (1 proces) | client-server (2 procesy) | client-server (4 procesy) |
|---|---|---|---|
| **Načtení dat s147** | 7,06s | 4,9s | 4,05s |
| **Načtení dat s183b** | 15,03s | 12,6s | 12,3s |
| **Použití filtru Clip na data s147** | 0,6s | 0,69s | 0,59s |
| **Použití filtru Clip na data s183b** | 2.04s | 2,05s | 1,59s |
| **Surface with Edges s daty s147** | 0,51s + 0,34s (send) | 0,54s + 0,4s (send) | 0,5s + 0,4s (send) |
| **Surface with Edges s daty s 183b** | 4.66s + 3.90s (send) | 4,7s + 3,8s (send) | 4,6s+3,7s (send) |

Při paralelních vizualizacích pracovaly všechny procesy a všechny měly stejné hodnoty časové náročnosti, proto je pro přehlednost v tabulce zobrazena jen jedna hodnota. U zobrazení dat v režimu Surface with Edges je nutné brát v potaz i čas, který byl potřeba pro zaslání výsledných dat klientovi. U ostatních operací se tato hodnota ve

výstupním souboru Timer Logu buď nezobrazovala, nebo byla skoro nulová. V tabulce je označena slovem send, ve výpisu souboru měla hodnotu vtkMPIMoveData.

U použití filtru Clip jsem z výstupu Timer Logu zjistil, že celou operaci prováděl při obou paralelních konfiguracích vždy pouze master. Dle mého názoru nebyla tolik časově a výpočetně náročná, aby ji dal svým sluhům. Hlavně z časového hlediska by se tento typ operace rozdělený mezi více nodů nejpíše nevyplatil. Museli bychom brát v potaz, jak rozdělení dat plus následné zaslání výsledku zpátky, tak i Ghost cells (popsáno v kapitole 4.1 Ghost levels), což by bylo časově náročnější. Oproti tomu u změny zobrazení na Surface with Edges pracovaly vždy všechny nody.

Z hodnot časové náročnosti při použití obou typů dat vidíme téměř 3s rozdíl při jejich načítání. To je způsobeno rozdělením dat do oddílů příslušným uzlům (sluhům). Výpočty trvaly u všech tří konfigurací téměř stejnou dobu, až na 0,5s rozdíl u použití filtru Clip na datech s183b. U ostatních operací se o žádném rozdílu mluvit nedá. Všechny hodnoty se pohybují v rozmezí několika desítek setin, což za zrychlení nepovažuji.

Můj předpoklad, který jsem si určil v průběhu této bakalářské práce, byl tedy mylný. S většími daty nebude paralelní vizualizace provedená na Hydře časově výhodnější, než s daty menšími.

Podle naměřených dat zobrazených v tabulkách 2 a 3 s výše zmiňovanými konfiguracemi vidíme, že paralelní vizualizace přináší jen minimální výhody s ohledem na časovou, nebo paměťovou náročnost výpočtu. Pouze při načítání dat ze serveru a použití filtru Clip na větších datech se vyskytl časový rozdíl mezi sériovým a paralelním výpočtem, při ostatních výpočtech jsou však naměřené hodnoty téměř identické. Větší rozdíl by se měl projevit při rozsáhlejších, časově náročnějších (v řádech dní) vizualizacích, s použitím většího množství filtrů.

# Závěr

Seznámil jsem se s problematikou vizualizace velkých dat a vyzkoušel si jednoduché vizualizace jak sériově, tak paralelně. Zjistil jsem architekturu a možnosti spuštění ParaView v paralelním módu a popsal jsem mechanismus výměny dat a komunikace mezi jednotlivými procesy. Zkompiloval jsem ParaView připravený pro paralelní vizualizace na systému CentOS a později danou problematiku přenesl i na školní výpočetní cluster Hydra. Pomocí MPI paralelizace jsem provedl paralelní vizualizace velkých dat, distribuovaných na jednotlivých uzlech Hydry. Porovnal jsem, jak využití systémové paměti klienta při konfiguraci standalone a client-server, tak i sériovou a paralelní vizualizaci s ohledem na časovou a paměťovou náročnost.

Výsledky měření systémové paměti klienta vyzněly jasně pro paralelní způsob vizualizace, jelikož se data nacházela přímo na výpočetním clusteru. Mezi sériovou client-server a paralelní client-server vizualizací nebyl s ohledem na paměťovou náročnost žádný rozdíl. Výpočetní cluster byl v obou případech vytížen stejně. Avšak s ohledem na časové vytížení bylo zjištěno zrychlení při načítání dat ve prospěch paralelního provedení.

Dané výsledky a rozdíly vizualizací musíme brát s ohledem na architekturu výpočetního serveru Hydra, kde je propojení clusterů pomalé a přenos dat mezi nimi tedy zkresluje výsledné hodnoty paralelní vizualizace. Při použití clusteru s rychlejší komunikací mezi jednotlivými nody by se měla zmenšit časová prodleva při předávání dat a paralelní vizualizace by dle mého názoru mohla být časově výhodnější.

# Použitá literatura

[1] Hydra. *Ústav nových technologií a aplikované informatiky.* [online]. [cit. 15. 3. 2015]. Dostupné z: <http://www.nti.tul.cz/cz/Hydra>

[2] Hydra/techdetails. *Ústav nových technologií a aplikované informatiky.* [online]. [cit. 15. 3. 2015]. Dostupné z: <http://www.nti.tul.cz/cz/Hydra/techdetails>

[3] Introduction to MPI and OpenMP. *Electrical and Computer Engineering Dept. Ryerson University.* [online]. [cit. 7. 5. 2015]. Dostupné z: <http://www.ee.ryerson.ca/~courses/ee8218/mpi_openmp.pdf>

[4] Message Passing Interface, *Wikipedia, The Free Encyclopedia.* [online]. [cit. 6. 10. 2014].

Dostupné z: <http://en.wikipedia.org/wiki/Message_Passing_Interface>

[5] Collective Communications with MPI. [online]. [cit. 7. 5. 2015]. Dostupné z: <http://static.msi.umn.edu/tutorial/scicomp/general/MPI/content3_new.html>

[6] Open MPI, *Wikipedia, The Free Encyclopedia.* [online]. [cit. 6. 2. 2015]. Dostupné z: <http://en.wikipedia.org/wiki/Open_MPI>

[7] MPICH, *Wikipedia, The Free Encyclopedia.* [online]. [cit. 6. 10. 2014]. Dostupné z: <http://en.wikipedia.org/wiki/MPICH>

[8] OpenMP. *openmp.org*. [online]. [cit. 7. 5. 2015]. Dostupné z: < http://openmp.org/wp/>

[9] ParaView. *Wikipedia, The Free Encyclopedia.* [online]. [cit. 15. 3. 2015]. Dostupné z: <http://en.wikipedia.org/wiki/ParaView>

[10] About ParaView. *ParaView.* [online]. [cit. 15. 3. 2015]. Dostupné z: <www.paraview.org/paraview/project/about.html>

[11] Users Guide Client-Server Visualization. *ParaView.* [online]. [cit. 6. 5. 2014].

Dostupné z: < http://www.paraview.org/Wiki/Users_Guide_Client-Server_Visualization>

[12] Large Scale Visualization with ParaView. kuleuven.be. [online]. [cit. 9. 4. 2015]. Dostupné z: <https://perswww.kuleuven.be/~u0016541/MHD_sheets_pdf/SC07_tut107_ParaView_Handouts.pdf>

[13] VERIFI Workshop-Hands ParaView Architecture. *Argonne blogs.* [online]. [cit. 19. 4. 2015]. Dostupné z: <https://blogs.anl.gov/verifi/files/2014/10/VERIFI_Workshop-Hands-on-Part-IIIc.pdf>

[14] ParaView – ParallelVisualisation. [online]. [cit. 18. 4. 2015]. Dostupné z: <http://goo.gl/wJRy0Y>

[15] CHILDS, Hank, Charles HANSEN a E. Wes BETHEL. *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Florida, USA: CRC Press, 2012. ISBN 9781439875728.

[16] ParaView : Build and Install. *ParaView.* [online]. [cit. 19. 4. 2015]. Dostupné z: <http://www.paraview.org/Wiki/ParaView:Build_And_Install>

[17] Paraview Roll for Rocks Cluster. [online]. [cit. 16. 1. 2015]. Dostupné z: <http://ie.archive.ubuntu.com/disk1/disk1/download.sourceforge.net/pub/sourceforge/s/sl/slurm-roll/addons/6.1/rolls/paraview/pb-paraview-roll.pdf>

[18] Running ParaView in parallel on the ELvis cluster. *surfsara.nl*. [online]. [cit. 16. 4. 2015]. Dostupné z: <https://surfsara.nl/documentation/parallel-paraview>