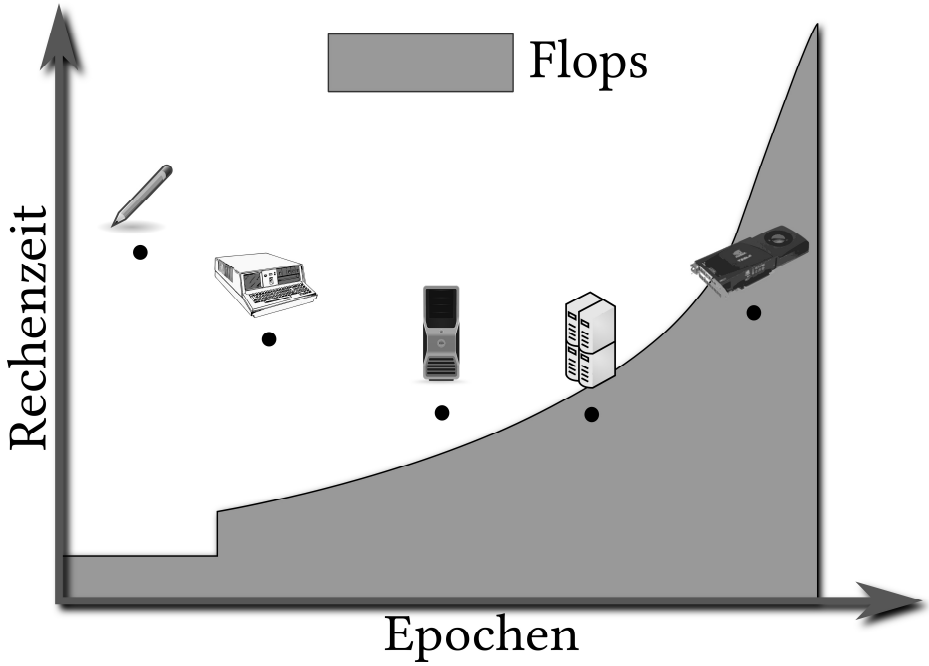
Aqua Urbanica 2012, 08.05.2012, IFAT ENTSORGA München
Siedlungsentwässerung im Wandel
Copyright © 2012, Schriftenreihe Fachgebiet Siedlungswasserwirtschaft
Band 32, TU Kaiserslautern (2012), S. D1 – D22

Vom Zeitbeiwertverfahren zum Mehrprozessorsystem - Über den Wandel der Methoden in der Kanalnetzberechnung

Gregor Burger, Manfred Kleidorfer, Wolfgang Rauch

Universität Innsbruck, Technikerstraße 13, 6020 Innsbruck, Österreich

Grafische Kurzfassung:



Key-Words Siedlungsentwässerung, Berechnung, Modellierung, parallel computing

1 Einführung

Anlagen und Methoden der Siedlungsentwässerung müssen zwei grundsätzliche Aufgaben erfüllen, das sind zum einen der „Schutz des Menschen vor der Natur“ und zum anderen der „Schutz der Natur vor dem Menschen“. Während die Entwässerungssysteme unter Trocken-

wetterbedingungen keine Probleme aufwerfen, stellen Starkniederschläge für beide Aufgaben die maßgeblichen Anforderungen dar. Die erste Aufgabenstellung umfasst daher die Konzeption und Dimensionierung des Kanalsystems in einer Art und Weise, dass Überlastungen des Systems (d.h. Überflutungen der urbanen Flächen) nur mehr sehr selten (z.B. weniger als einmal pro Jahr) auftreten. Eine vollständige Vermeidung ist – analog zum Hochwasserschutz – weder technisch möglich noch ökonomisch sinnvoll.

Für den Schutz der Gewässer ist die Siedlungsentwässerung dergestalt zu konzeptionieren, dass bei Regenwetter eine möglichst geringe Stofffracht über die Mischwasserentlastung in die Gewässer geleitet wird, bzw. ein möglichst hoher Prozentsatz an der Kläranlage behandelt wird. Die vollständige Behandlung des Abflusses unter Trockenwetterbedingungen an der Kläranlage stellt kein Problem der Siedlungsentwässerung dar und wird vorausgesetzt.

Die beiden primären Ziele der Bemessung lauten daher:

- Bemessung des Kanalsystems im Hinblick auf Überflutung
- Bemessung des Kanalsystems im Hinblick auf entlastete Stofffrachten

Für diese beiden Dimensionierungsaufgaben wurden historisch empirische Beziehungen und einfache Berechnungsformeln verwendet, es handelte sich also um so genannte „Papier und Bleistift“ Berechnungen. In den letzten beiden Jahrzehnten wurden aber vermehrt numerische Modelle verfügbar, die eine detaillierte Simulation der Dynamik der Vorgänge bei Regenwetter ermöglichen. Dies führte auch zu einem radikalen Wandel in den Richtlinien, indem seit kurzem die numerische Simulation als Stand der Technik im Entwurf von Kanalisationssystemen vorgeschrieben wird. Mit zunehmender Komplexität der Modelle (immer detailliertere Beschreibung der Prozesse) sind zunehmend auch Methoden der Systemanalytik immer wichtiger (z.B. Sensitivitätsanalysen, Unsicherheitsanalysen etc.). Solche Methoden erfordern wiederum aufwändige Simulationen (z.B. lange Zeiträume, Monte-Carlo Simulationen) und oft ist die Rechenzeit der Modelle ein limitierender Faktor in der Modellierung. Ein möglichst gutes Ausnützen der Leistungsfähigkeit moderner

Computer ist heute also das Ziel. Dafür sind jedoch entsprechende Formulierungen der Modelle notwendig.

2 Block und Bleistift

Durch Arbeiten von [Mulvaney 1851] in Irland und [Kuichling 1889] in den USA stand bereits zu Beginn des modernen Kanalbaus mit dem Fließzeitverfahren eine Methode zur quantitativen Berechnung des Abflusses unter Regenwetterbedingungen zur Verfügung. Dieser klassische Ansatz der Listenrechnung wird - mit Modifikationen - bis heute im Entwurf von Kanalisationen verwendet, allerdings in den letzten 20 Jahren zunehmend ersetzt durch numerische Modelle.

Tabelle 1: Beispiel einer Listenrechnung (Zeitbeiwertverfahren) für Abflussermittlung

Nr.	$A_{E,k}$ (ha)	ψ (-)	A_{red} (ha)	ΣA_{red} (ha)	L (m)	v (m/s)	t (min)	Σt (min)	φ (-)	Q_R (l/s)
S7	8,7	0,3	2,6	2,6	212	1,0	3,5	8,5	1,64	640
S6	11,0	0,3	3,3	3,3	278	1,0	4,6	9,6	1,64	812
S5	5,2	0,5	2,6	8,5	192	1,0	3,2	12,8	1,43	1823
S4	6,2	0,5	3,1	11,6	191	1,0	3,2	16,0	1,25	2175

3 Computerrevolution

Die Entwicklung von numerischen Berechnungsmethoden wurde maßgeblich von der US-EPA beeinflusst, die bereits 1969 ein Konsortium unter Führung von Metcalf und Eddy mit der Erstellung der Software SWMM (Stormwater Management Model) beauftragte. Seit der ersten Version im Jahre 1971 wurde dieses Berechnungswerkzeug bis heute

kontinuierlich weiterentwickelt. Mit SWMM 5 steht ein hydrodynamisches Berechnungsmodell zur Simulation von Abflüssen und Stofffrachten in Kanalsystemen zur Verfügung, das aufgrund des Status als „public domain software“ eine sehr weite Verbreitung aufweist.

Ausgehend von diesem Modell wurden seit den 80er Jahren auch in Europa verschiedene Modelle und Softwareprodukte für die dynamische Simulation des Abflusses entwickelt. Bekannt und kommerziell erfolgreich sind hier vor allem Hystem-Extran und Kanal++ aus Deutschland, Mouse bzw. MikeUrban aus Dänemark und InfoWorks aus England.

Im Vergleich zur Abflussberechnung wurde die Frage des Gewässerschutzes im deutschsprachigen Raum historisch direkt an die Bemessung von Mischwasserentlastungbecken geknüpft. D.h. es wurden empirische Regeln für die Dimensionierung der Becken entwickelt, welche vereinfacht das Systemverhalten und die Redundanz des Vorfluters berücksichtigen. Diese simplen Bemessungsformeln wurden nur langsam durch numerische Berechnungsmethoden ersetzt – in Österreich z.B. erst mit der Neufassung des ÖWAV Regelblattes 19 „Richtlinien für die Bemessung von Regenentlastungen in Mischwasserkanälen“ im Jahre 2007.

Die numerische Berechnung der Stofffrachten bedingt die kontinuierliche Simulation des Geschehens über einen relevanten Zeitraum von ca. 10 Jahren. Obwohl hierzu grundsätzlich auch die vorhin erwähnten hydrodynamischen Modelle in der Lage sind, wurden wegen der geringeren Rechendauer einfachere hydrologische Modelle entwickelt. Erste Softwareprodukte entstanden meist in Zusammenhang mit Dissertationen z.B. entwickelte [Paulsen 1987] das Werkzeug KOSIM (Uni Hannover) und [Johansen und Hygiejne 1985] SAMBA (TU Dänemark). In den folgenden Jahren wurden weitere Software Werkzeuge entwickelt – meist auch in direktem Zusammenhang mit den landesspezifischen Regelwerken: z.B. REBEKA in der Schweiz [Rauch, Krejci, und Gujer 2002] und KAREN für Österreich [Rauch und Kinzel 2007].

3.1 Modellierung

Ein Modell ist eine schematische, mathematische Beschreibung von physikalischen Zusammenhängen mit dem Zweck komplexe Vorgänge fassbar zu machen, Zusammenhänge zu erklären und Vorhersagen zu machen. In der Siedlungsentwässerung dienen Modelle als Planungswerkzeug für die Bemessung und den Nachweis der Leistungsfähigkeit unterschiedlicher Infrastruktureinrichtungen, wie beispielsweise zur Dimensionierung von Kanalisationssystemen, Mischwasserüberlaufbauwerken aber auch von Kläranlagen, Infiltrationsanlagen usw.

Niederschlag ist die treibende Kraft in allen Simulationsmodellen der Siedlungsentwässerung. Ausgehend von gemessenen oder statistisch generierten Niederschlagsdaten in hoher zeitlicher Auflösung von z.B. 5 Minuten wird der Regenabfluss aus einem Einzugsgebiet berechnet (Niederschlag-Abfluss Modellierung) und in weiterer Folge der Verlauf der Welle in den Kanalrohren (Transportprozesse). In den Rohren können sich Wellen durch Zusammenflüsse aus verschiedenen Einzugsgebieten überlagern und damit ansteigen oder durch eine Dämpfung abschwächen. Welche Effekte hier berücksichtigt werden, hängt von den verwendeten Transportgleichungen ab.

In der Niederschlags/Abfluss Berechnung werden üblicherweise die beiden Prozesse Abflussbildung (Wie viel des gefallenen Niederschlags kommt zum Abfluss?) und Abflusskonzentration (Welche Form hat die Abflusswelle aus dem Einzugsgebiet?) getrennt gesehen. In der Berechnung der Transportprozesse (Wie fließt die Welle in den Haltungen ab?) wird üblicherweise zwischen zwei Berechnungsansätzen „hydrodynamisches“ und „hydrologisches“ Modell unterschieden. Beide letztgenannten Ansätze unterscheiden sich nur darin wie detailliert die Transportvorgänge im Kanal berechnet werden. Die verschiedenen Untermodelle (Abflussbildung, Abflusskonzentration, Kanaltransport) werden miteinander kombiniert, wobei durchaus unterschiedliche Ansätze (komplex und einfach) miteinander verknüpft werden können.

Werden Stofffrachten berücksichtigt müssen natürlich weitere Prozesse beachtet werden. Dies ist erstens die Entstehung der Verschmutzung aus den drei Quellen Trockenwetterabfluss, Niederschlag und mobilisier-

te Stoffe im Kanal und zweitens der Transport der Stoffe in der fließenden Welle. Der zweite Prozess kann für gelöste Stoffe mit der bekannten Advektions-/ Dispersionsgleichung - zumindest theoretisch - gut bestimmt werden, während der Transport von partikulären Stoffen bis heute ein Problem darstellt.

4 Mehrkernrevolution

In den letzten 30 Jahren wurden bemerkenswerte Performancesteigerungen im Bereich der Chipherstellung erzielt. Es war möglich immer mehr Performance zu erlangen indem die Taktraten erhöht, die Ausführungen optimiert und die Caches erweitert und verbessert wurden. Dies führte dazu, dass Software mit jedem Jahr, ohne weiteres beisteuern der Entwickler, immer schneller wurde. Moores Gesetz besagt, dass die Anzahl der Transistoren alle 16 Monate verdoppelt wird [Moore 1998]. Dies führte dazu, dass nach dem 1Ghz Prozessor der 2Ghz Prozessor folgte und wir schließlich und endlich bei Taktraten von 3Ghz angelangt sind.

Verdoppelung alle 16 Monate bedeutet exponentielles Wachstum. Seit 2005 sollten unter unseren Schreibtischen 10Ghz Rechner ihre Werke verrichten. Wie jedes exponentielles Wachstum hatte auch das Streben nach immer höheren Taktfrequenzen ein Ende. Der Grund für den abrupten Abriss der Taktfrequenz-Steigerungen sind physikalische Grenzen. Zum Beispiel ist die Lichtgeschwindigkeit eine absolute Größe auf die wir keinen Einfluss haben. Probleme mit denen die Chiphersteller zu kämpfen haben sind, dass die Prozessoren zu heiß werden, zu viel Strom verbrauchen und durch die immer kleineren Auflösungen in der Prozessorherstellung immer mehr Probleme mit Fehlerstrom auftreten [Herb Sutter 2005].

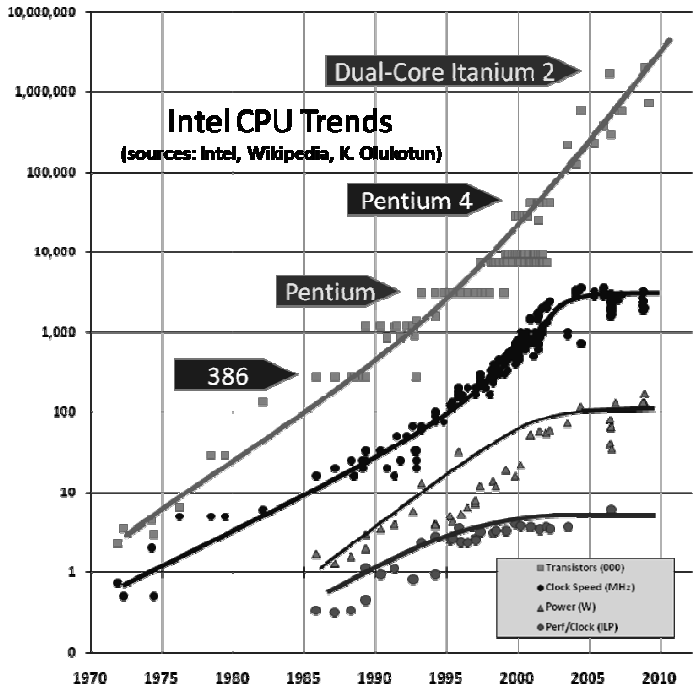


Abbildung 1: Die Entwicklung der Transistoren, Taktfrequenzen, Leistungsaufnahme und Geschwindigkeit pro Takt (Quelle: Herb Sutter, 2005)

Das Versprechen nach einer Verdopplung der Transistoren alle 16 Monate muss gehalten werden. Der große Druck der Chiphersteller ließ sie jedoch ein anderes Versprech brechen: „Alle 16 Monate wird ohne Änderung an der Software genau diese doppelt so schnell“. Die Lösung der Hersteller war es auf die Erhöhung der Taktfrequenzen und all ihrer Probleme zu verzichten und auf Mehrkernsysteme zu setzten. Wie in Abbildung 1 zu sehen ist, begann sich Anfang 2004 ein Plateau in den Taktfrequenzen abzubilden. Die Transistoren stiegen jedoch weiter, was der Einführung der Mehrkernrechner zu verdanken ist [Olukotun u. a. 1996].

Würde man sich eine weitere, fiktive Kurve in dem Diagramm in Abbildung 1 vorstellen, welche die Geschwindigkeit der Computerprogramme beschreibt, so hätte diese Kurve eine ähnliche Ausprägung wie jene Kurve der Taktfrequenz. Es würde sich ein deutliches Plateau ausbilden. Der Grund hierfür wurde bereits erwähnt; es wurde das Versprechen gebrochen, dass Programme ohne Änderungen an den Fortschritten der Prozessoren profitieren. Herb Sutter tituliert all jene Probleme sehr gut in seinem Artikel „The free lunch is over“ in dem er einen fundamentale

Wende in der Softwareentwicklung in Richtung parallele Datenverarbeitung beschreibt [Herb Sutter 2005].

4.1 Parallele Datenverarbeitung

Die parallele Datenverarbeitung (PC) kommt aus dem High Performance Computing (HPC) Bereich, wo schon seit mehreren Jahrzenten die Aufteilung von Problemen auf mehreren Prozessoren/Computern ein bekanntes Problem beschreibt. Durch die Mehrkernrevolution drang PC wohl oder übel auch in den Bereich des General Purpose Computing (GPC). Wohingegen man sich im HPC Bereich schon seit mehreren Jahren mit den Problemen des PC auseinandersetzte kam die Mehrkernrevolution im GPC unangemeldet über Nacht. Programme welche nicht an diese neuen Umstände angepasst werden, können im besten Fall mit einer stagnierenden Performance rechnen. Will man jedoch die brachliegenden Ressourcen der neuen Prozessoren nutzen so sind teils tiefgreifen Änderung an den Softwareprodukten nötig. Dieser Abschnitt soll einen kleinen Einblick geben was es bedeutet Software für parallele Rechner zu entwickeln, warum es ungleich schwerer ist, mit welchen Problemen zu rechnen ist und was die Zukunft noch mit sich bringt.

Es gibt zwei Möglichkeiten wie man die Ressourcen von Mehrkernrechner ausnutzten kann. Einerseits die besprochenen PC in der es möglich ist, dass ein und derselbe Algorithmus eine große Menge an Daten parallel abarbeitet. Auf der anderen Seite besteht die Möglichkeit mehrere Algorithmen gleichzeitig auszuführen welche auf dieselben oder unterschiedliche Daten zugreifen (im englischen concurrency genannt). Ein Unterschied der sich meist im Grad der Parallelisierung niederschlägt. Der Parallelisierungsgrad beschreibt die Anzahl an Prozessoren (können auch Kerne oder Rechner sein; Einfachheitshalber aber nur Prozessoren) welche gleichzeitig bedient werden können. Je höher der Parallelisierungsgrad desto besser kann die aktuelle bzw. zukünftige Hardware ausgenutzt werden. Im PC sind meist die Anzahl der Daten ein Limit, jedoch sind im generellen Fall genügend Daten vorhanden. Im anderen Fall ist der Parallelisierungsgrad abhängig von den Algorithmen die gleichzeitig laufen können, welche meist sehr beschränkt sind.

In beiden Fällen hat man jedoch die gleichen zugrunde liegenden Probleme. Die größten Probleme die bei der Entwicklung von parallelen Anwendungen auftreten passieren, wenn mehrere Ausführungsstränge (Threads) gleichzeitig auf dieselben Daten zugreifen. Diese sogenannten Wettlaufsituationen (im engl. Race Condition) können zu korrupten Daten, Abstürzen oder undefiniertem Verhalten führen. Die Erkennung solcher Probleme ist oft sehr schwer, da diese nicht immer auftreten müssen. Das Aufspüren oder schon im Vorfeld verhindern solcher Probleme erfordert langjährige Erfahrung mit nebenläufigen Algorithmen. Abhilfe schaffen bei solchen Problemen Synchronisierungen. Beispiele für Synchronisierungsmethoden können Locks, Mutexes, Monitore, Barriers oder Semaphoren sein, welche einen kritischen Abschnitt im Code davor bewahren von mehreren Threads gleichzeitig durchlaufen zu werden oder es erlauben Threads aufeinander zu warten und damit miteinander zu kommunizieren [Grama, Gupta, und Karypis 2003].

Hat man es geschafft allen Datenkorruptionen auszuweichen kann es noch zu Deadlocks, Livelocks oder Starvations kommen. Ein Deadlock beispielsweise ist ein Zustand, indem sich zwei Threads unwiederbringlich am weiterarbeiten hindern, indem sie auf Locks warten die bereits vom anderen Thread gehalten werden. Deadlocks sind ähnlich unangenehm zu finden und zu behandeln wie Wettlaufsituationen. Denn anders als bei sequentiellen Programmen wo Abstürze sehr gut reproduzierbar sind, können Fehler in parallelen Programmen oft Jahre schlummern, bevor sie ausgelöst werden. Die meisten Fehler werden durch Ausführungstests oft nicht erkannt, sorgfältiges Programmieren und Entwickeln im Vorhinein ist daher essentiell für eine funktionierende parallele Anwendung.

Sind die letzten Locking Probleme gelöst, die ein korrektes Ausführen des Programms verhindern, so kommt es schließlich und endlich zu der entscheidenden Frage: „Ist das Programm schneller?“. Locking kann grob oder fein granuliert sein. Beim groben Locking (z.B.: Zugriff auf alle Daten für nur einen Thread erlauben) besteht die Gefahr, dass man andere Threads an der Ausführung hindert. Beim fein granulierten Locking (z.B. wenn jede kleinste Dateneinheit gelockt wird) läuft man Gefahr, dass der Mehraufwand der durch Schließen, Lösen und Warten auf Locks entsteht die eigentliche Ausführung überragt. Das Programm oder

der Algorithmus kann trotzt des Parallelisierens langsamer sein. Wie bereits erwähnt ist für PC langjährige Erfahrung und ein ungleich größerer Aufwand nötig als für die klassische sequentielle Programmierung, wo es nicht darum ging hunderte von Threads davon zu hindern Amok zu laufen.

Doch bevor man sich überhaupt Gedanken über die korrekte Ausführung von parallelen Algorithmen macht gilt es zu überlegen, wie man ein Programm/einen Algorithmus auf mehrere Threads aufteilt. Die Entscheidung welche Teile des Codes parallel laufen und wie viele davon, beeinflusst wiederum das Locking und damit die Performance der Anwendung. Eine sehr gute Kenntnis des zugrunde liegenden Algorithmus ist absolut essentiell. Nur in den wenigsten Fällen ist es bei bestehenden Programmen möglich eine Parallelisierung durchzuführen ohne das tiefgreifende Umbauten am Code nötig sind. Teilweise müssen sogar ganze Codeteile umgestellt oder ein komplett neues Design entwickelt werden. Sehr viele Algorithmen lassen sich überhaupt nicht oder nur zu einem sehr geringen Grad parallelisieren. Ist bei solchen Problemen eine Performancesteigerung nötig so muss man sich nach parallelen Algorithmen umsehen, welche ein gegebenes Problem lösen können.

Die Suche nach genau jenen Algorithmen, die sehr gut auf parallelen Architekturen funktionieren ist ein wichtiges Thema und wird in Zukunft immer mehr an Bedeutung finden. Mehrkernprozessoren mit 8, 16, 24 Kernen sind nicht das Ende der Fahnenstange. Vielkernprozessoren mit mehreren hundert Prozessoren sind bereits ein essentieller Bestandteil im HPC und dringen immer weiter in den GPC Bereich. Die bekanntesten Vielkernprozessoren kommen aus der Spielebranche, also eigentlich aus dem Heim PC Bereich und beschleunigten in den letzten Jahren 3D Spiele, welche mehrere Millionen Polygone 24-mal pro Sekunden auf den Bildschirm brachten. Diese Grafikkarten (Graphical Processing Units GPU) finden im Scientific und HPC immer mehr Anwendung. Ein Algorithmus welcher eine GPU zu beschleunigen vermag muss jedoch einen erheblichen Parallelisierungsgrad aufweisen. Die einzelnen Kerne der GPUs sind meist wesentlich langsamer als ein Kern eines gängigen CPUs. Die Einfachheit der einzelnen Kerne erlaubt es jedoch weit mehr von ihnen auf einem Chip unterzubringen. Die schiere Anzahl der Kerne und ein massiv paralleles Problem erlauben dann Performancesteige-

rungen die jenseits von Mehrkernprozessoren liegen. Jedoch sind nur sehr wenige Probleme für solche Vielkernprozessoren geeignet.

4.2 Parallele Simulationswerkzeuge

In diesem Abschnitt werden die bisher erreichten Ziele in der Parallelisierung von Abwassersimulationswerkzeugen beschrieben. Es soll gezeigt werden was möglich ist, wie es erreicht wurde, und wie viel von parallelisierten Simulationstools zu erwarten ist. Bei den folgenden Projekten waren unterschiedliche Herangehensweisen von Nöten, welche aber ein interessantes Licht auf die Parallelisierung selbst werfen.

4.2.1 CityDrain3

CityDrain3 ist weniger ein spezifisches Programm als eher eine allgemeine Entwicklungsumgebung zur zeitlich dynamischen Simulation von urbanen Entwässerungssystemen. Die Software ermöglicht die Berechnung von Wasser- und Stoffflüssen in frei programmierbaren Modulen innerhalb einer Baumstruktur. Physikalische und/oder biochemische Prozesse können innerhalb der Module mit beliebiger mathematischer Formulierung abgebildet werden. Aufgrund der vorgegebenen Baumstruktur werden die Wasser- und Stoffflüsse aber mit konzeptionellen Modellen berechnet, d.h. die Abflüsse werden mit konzeptionellen (hydrologischen) Modellen ermittelt. CityDrain3 basiert auf den mathematischen und modelltechnischen Konzepten von CITYDRAIN. Es wurde jedoch von Grund auf neu entwickelt, um die bestmögliche Performance zu erreichen und um die schwerfällige und recht unhandliche Matlab-Umgebung zu verlassen. Der neue Code sollte des weiteren leicht erweiterbar und in andere Produkte integrierbar sein [Achleitner, Moderl, und Rauch 2007].

Die Erweiterbarkeit von CityDrain3 wurde durch ein stark modulares, objekt-orientiertes Design erreicht. Abbildung 2 zeigt die Zusammenhänge und Eigenschaften der Kernclassen in CityDrain3. Im Zentrum steht dir Klasse *INode.* Sie beschreibt ein Element des integrierten Abwasserkreislaufes. Diese Elemente können entweder konkret sein, wie etwa ein Kanal, CSO oder eine Kläranlage oder auch fiktiv, wie etwa ein Dateiausgabeknoten oder ein Regenleser. Verbunden werden die Knoten von

sink nach *source* über *INodeConnection* und in der Klasse *Model* zu einer vollständigen Abwassersystembeschreibung zusammengefasst. Das Interface *ISimulation* ist für den Ablauf der Simulation verantwortlich.

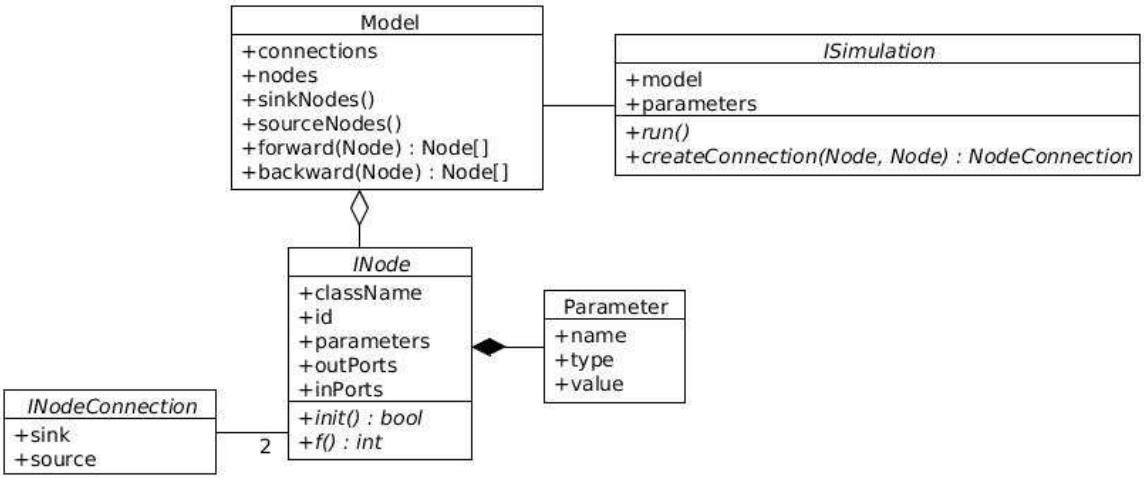


Abbildung 2: Kernklassen von CityDrain3

Das sehr ausgeklügelte Design ermöglicht es Knoten sowohl in der nativen Programmiersprache (C++) von CityDrain3 als auch über Skriptsprachen (Python) hinzuzufügen. Neue Knoten können ohne Änderung an dem Kern oder der grafischen Oberfläche von CityDrain3 geladen und verwendet werden. Ein Beispiel eines Systems mit vier Knoten und drei Verbindungen ist in Abbildung 3 zu sehen.

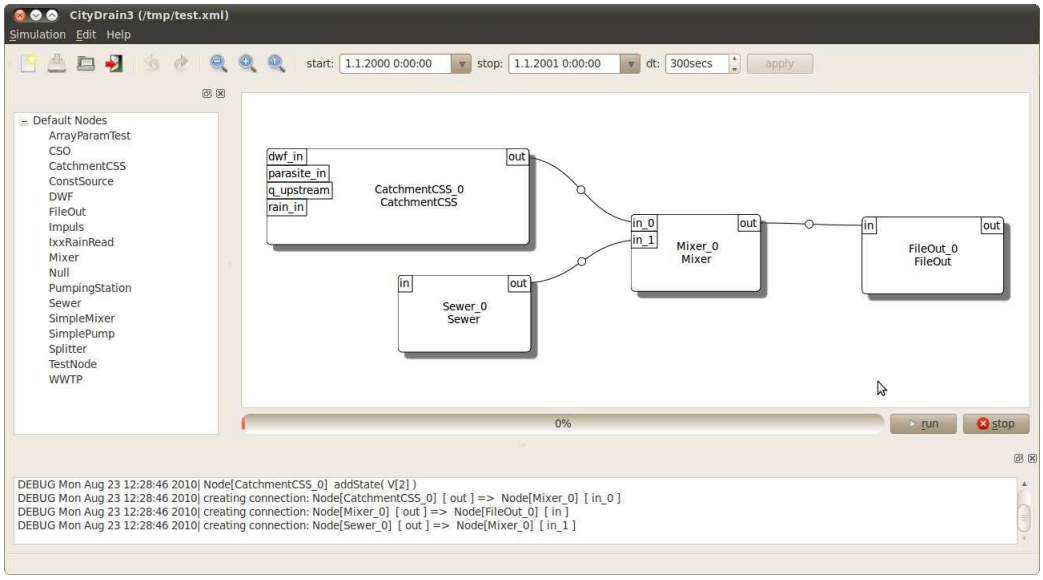


Abbildung 3: Die grafische Oberfläche von CityDrain3

Über die Abstraktion des *ISimulation* interface war es möglich mehrere Strategien für Simulationsabläufe zu definieren. Dies mag auf den ersten Blick etwas übertrieben sein, da Simulation normalerweise immer gleich ablaufen (pro Zeitschritt werden alle Knoten von Quellen bis zur Senke abhängig ihrer Verbindungen abgearbeitet), führte jedoch dazu, dass mehrere Parallelisierungsstrategien getestet und entwickelt worden sind.

Die drei Strategien sind Flow Parallel Strategy (FPS) Pipelined Parallel Strategy (PPS) und Ordered Parallel Strategy (OPS). Bei FPS werden pro Quelle, in großen Systemen gibt es meist viele Quellen die ein Catchment beschreiben, ein Thread gestartet. Dieser Thread folgt der Abarbeitung der Knoten flussabwärts. Bei Knoten mit mehreren Inputverbindungen muss darauf gewartet werden bis alle Knoten flussaufwärts, welche Verbindungen zu diesem Knoten besitzen, vollständig abgearbeitet sind. Ein Vorteil dieser Strategie ist, dass die Implementierung sehr einfach und überschaubar ist. Der große Nachteil ist jedoch, dass der Parallelisierungsgrad von der Anzahl der parallelen Stränge im Wassersystem limitiert ist. Der schlimmste Fall wäre, wenn ein System nur eine Quelle besitzt und alle Knoten innerhalb des Systems nur mit einem Knoten flussauf- und -abwärts verbunden sind [Burger u. a. 2010].

Mit der PPS Strategie wollte man die Nachteile von FPS beseitigen und die Tatsache ausnutzen, dass mehrere Zeitschritte gleichzeitig gerechnet werden können. Die Voraussetzung, dass ein Knoten für einen bestimmten Zeitschritt gerechnet werden kann ist, dass all jene Knoten die Inputs abgeben bereits in diesem Zeitschritt gerechnet worden sind. Bei PPS ist ein Thread für genau einen Zeitschritt verantwortlich, z.B. ist der erste Thread für Zeitschritt null zuständig. Jeder Thread bekommt einen Pool mit Referenzen zu allen Knoten und versucht diese für diesen Zeitschritt auszuführen. Knoten, welche die bereits erwähnten Voraussetzungen erfüllen werden ausgeführt und aus dem Pool entfernt. Ist der Pool leer wird dem Thread ein neuer Zeitschritt zugeteilt. Wie bereits erwähnt ist es mit dieser Strategie möglich auch sehr lineare Systeme zu parallelisieren in dem man die Zeitschritte durch das System pipelined. Der große Nachteil ist jedoch, dass die erste Umsetzung dieser Strategie zu hohe Wartezeiten durch eine schlechte, zufällige Auswahl an Knoten, welche als nächstes probiert werden aufwies [Burger u. a. 2010].

Um eine zufällige Auswahl zu vermeiden wurden die Knoten in der OPS Strategie topologisch sortiert. Diese Sortierung ermöglicht es, dass eine Abarbeitung in dieser Reihenfolge alle Voraussetzungen gezwungenermaßen erfüllt sind. Der Pool wurde zu einer Warteschlange und ein Thread ist dann fertig, wenn die topologisch sortierte Warteschlange leer ist. Jegliche Überprüfung auf erfüllte Voraussetzungen für die Abarbeitung der Knoten ist damit überflüssig und die Wartezeiten werden auf ein Minimum reduziert [Burger u. a. 2010].

Dies schlägt sich auch auf die Performancetests nieder. Abbildung 4 und Abbildung 5 zeigen die Speedups von zwei unterschiedlichen Inputsystemen. Das erste System ist die alpine Stadt Innsbruck. Es entspricht somit einem Realsystem mit 120000 Einwohnern. Das Realsystem weist einen starken Baumcharakter auf. Das zweite System ist ein worst-case scenario für die FPS Strategie, ein lineares System mit 1000 Knoten in Serie geschaltet. In beiden Fällen sind zwei Peaks zu sehen, bei vier und bei acht Threads. Diese Peaks entstehen aus der Konfiguration des Testsystems auf dem die Benchmarks ausgeführt wurden, welches einen Quad-Core i7 von Intel mit vier hypergethreadedeten Kernen besitzt. Es ist also sehr schön zu sehen, dass ein gewisser Mehraufwand durch das Hyperthreading entsteht, dieser jedoch mit steigender Anzahl von Threads wieder kompensiert werden kann. In beiden Fällen erzielt die OPS Strategie den höchsten Speedup von 3.9 bis 4.0.

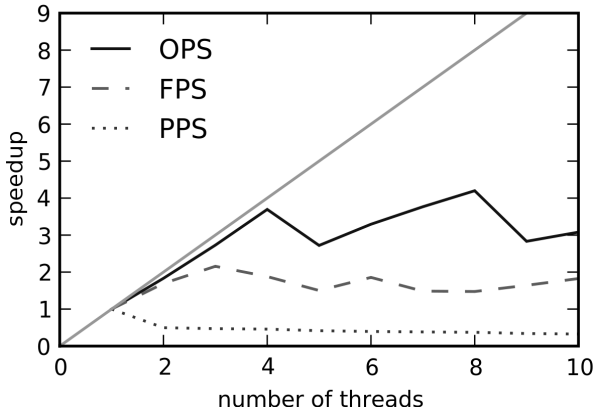


Abbildung 4: Innsbruck (Quelle: Burger u. a. 2010)

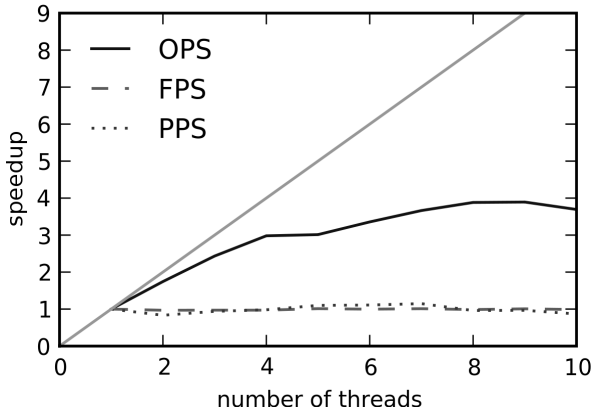


Abbildung 5: Sequentielles System (Quelle: Burger u. a. 2010)

4.2.2 SWMM

Das Stormwater Management Model (SWMM) ist ein hydrodynamisches Model um Regenfall und Regenabfluss zu berechnen. Die Regenabflusskomponenten bestehen aus Einzugsgebieten auf denen es abregnet und ein Abfluss gebildet wird. Die hydraulische Wasserableitung besteht aus einem System an Kanälen, Rohren, Tanks, Wasseraufbereitunsanlagen und Drosseln. SWMM wurde zuerst im Jahre 1971 entwickelt und war in Fortran geschrieben. Die jetzige Version ist 5.0 und wurde komplett neu in C entwickelt. SWMM wurde von der US-EPA entwickelt und ist somit in öffentlichem Besitz. Dieser Umstand trug zu der sehr weiten Verbreitung und Verwendung von SWMM bei [Huber 1995].

Die mit Abstand am meisten verwendete Routing Methode in SWMM ist die volle dynamische Welle. Sie basiert auf den von Saint Venant entwickelten Flachwassergleichungen. Diese Gleichungen werden mittels der finiten Elemente Methode diskretisiert und in einen Gleichungslöser eingebaut. Der Rechenaufwand dieser Gleichungen ist gegenüber dem konzeptionellen Modell von CityDrain3 ungleich größer. Jedoch wurde an den grundlegenden mathematischen Modellen und deren Umsetzung in SWMM schon seit mehr als 40 Jahre gefeilt und optimiert. Eine weitere Performancesteigerung in SWMM durch Parallelisierung ist daher sehr attraktiv.

Durch den bereits bestehenden und sehr gut optimierten Code von SWMM war eine komplett andere Vorangehensweise nötig als das im Fall von CD3 war. Desweiteren wird SWMM in sehr vielen kommerziellen sowie freien Produkten verwendet. Um die Verwendung der parallelen Version voranzutreiben war geplant die Änderungen an dem Code so gering wie möglich zu halten.

Für das Projekt waren mehrere Zyklen geplant in denen zuerst mittels Profiling diejenigen Codestellen gefunden wurden, an denen am meisten Rechenzeit verbraucht wird. Auf Grund der Unbekanntheit des Codes war eine sehr lange Code-Review Phase eingeplant in der die betreffenden Codeteile auf Parallelisierung untersucht wurden. Nach der Implementierung folgt dann eine Überprüfung des Codes auf parallele Programmierfehler und erneute Performancetests, welche einen weiteren Zyklus begannen.

Mit dieser Methode konnte erreicht werden, dass durch sehr wenige Änderungen an dem Code, sehr erheblichen Beschleunigungen am Rechenkern erzielt werden konnten. Der Aufwand dieser Änderungen war jedoch durch den Komplexitätsgrad und der gediegen gewachsenen Codebasis sehr zeitintensiv.

Es wurden mehrere Inputsysteme getestet, um zu sehen wie sich die Größe des Inputsystems auf den Algorithmus auswirkt. Zwei Beispiele der größeren Systeme sind in Abbildung 6 und Abbildung 7 zu sehen. Die Großstadt ist eine Hauptstadt in Österreich mit 120.000 Einwohnern wohingegen das zweite System nur 12.000 Einwohner auffasst.

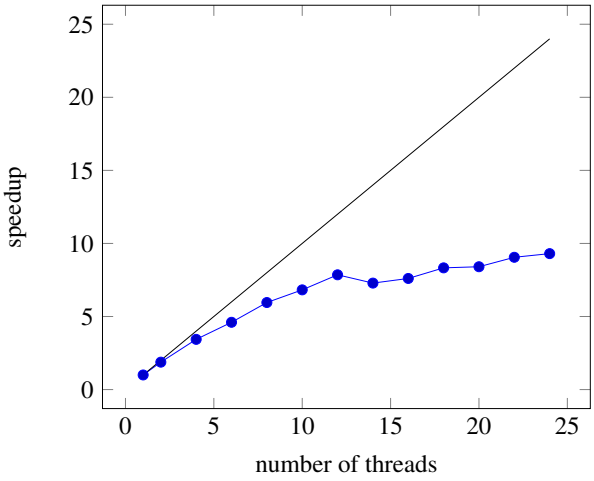


Abbildung 6: Großstadt

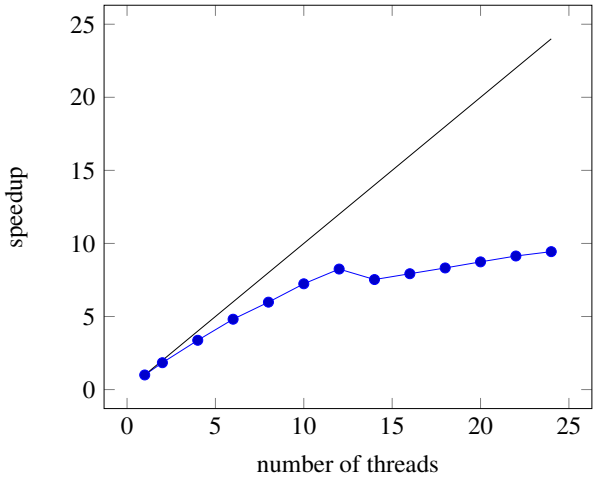


Abbildung 7: Ländliche Stadt

Ebenso wie in den CityDrain3 Benchmarks kann auch wieder in diesen Benchmarks ein Knick in der Kurve gesehen werden. Diesmal ist dieser Knick allerdings bei zwölf anstatt bei vier. Der Grund hierfür ist, dass einige Zeit zwischen den Benchmarks vergangen ist und diese Benchmarks auf einem Dual-Socket Sechskern System gelaufen sind. Dieser Rechner besitzt somit zwölf Kerne, mit Hyperthreading belaufen sich dann die Threads auf 24. Bei 24 Threads erholt sich der zusätzliche Aufwand des Hyperthreadings und der Speedup beläuft sich auf 9.3. SWMM mit Parallelisierung ist also rund zehn mal schneller auf einem Zwölfkern Rechner. Die Laufzeit der Großstadt ging dabei von vier Minuten auf nur 26 Sekunden herunter.

Bei einem sehr kleinen System, welches getestet wurde, um zu sehen ob das Threading einen negativen Einfluss hat, konnte eine Erhöhung

des Speedups bis acht Threads gesehen werden, diese ging jedoch zurück, blieb aber immer noch unter der Laufzeit einer sequentieller Ausführung. Demnach hat die Parallelisierung, auch bei kleinen Systemen, auf denen der Overhead im Vergleich zur restlichen Laufzeit groß ist, keinen negativen Einfluss.

5 Zukunft

Wie bereits erwähnt gehört die Zukunft den Vielkernprozessoren unter denen sich auch Hybride wie etwa der Cell Prozessor von IBM [Pham u. a. 2005] oder die Larabee Architektur von Intel [Seiler u. a. 2008]. Beide Prozessoren haben mehrere Cores, die sehr stark den Kernen heutiger CPUs ähneln, besitzen jedoch noch einige Streamingprozessor die sehr stark in die Richtung von GPU Kernen gehen. Diese zusätzlichen Kerne sind für die Ausführung von Grafikoperationen gedacht, können aber ähnlich wie GPUs dazu verwendet werden spezielle, hoch parallele Algorithmen auszuführen. Die Programmierung solcher Hybriden und der GPUs stellt die Softwareentwicklung vor weitere Probleme und es gilt sich in diesem Dschungel zurecht zu finden [H. Sutter 2011].

Genau jene Algorithmen, welche sich für diese Architekturen gut eignen gilt es zu finden und für die Siedlungsentwässerung zu evaluieren. An der Universität Innsbruck / Arbeitsbereich für Umwelttechnik wird gerade einer dieser Ansätze für den Einsatz in der Siedlungsentwässerung überprüft. Das Verfahren nennt sich Smoothed Particle Hydrodynamics (SPH) und stammt aus der Astrophysik und wurde dort von [Gingold und Monaghan 1977] gleichzeitig mit [Lucy 1977] entwickelt, um kollidierende schwarze Löcher und das Entstehen von Galaxien zu simulieren.

SPH ist eine Lagranges Diskretisierungsverfahren, entgegengesetzt zu den eulerschen Diskretisierungsverfahren wie etwa die Finiten Elementen oder Finiten Volumen. Der Unterschied besteht darin, dass der Betrachter einem Flüssigkeitspaket folgt [Batchelor 2000; Lamb 2010]. Dadurch ist man von den gängigen Nachteilen der gitterbasierenden Methoden befreit. Diese Flüssigkeitspakete werden Partikel genannt und können sich im Raum frei bewegen. Partikel interagieren mittels einer

Glättungskurve mit den benachbarten Partikeln [Gingold und Monaghan 1977].

Schon bald war klar, dass SPH für mehr einzusetzen ist als für Gassimulationen im All. Sowohl die Filmindustrie als auch Wissenschaftler in der Festkörpermechanik und der numerischen Strömungsmechanik fanden Gefallen an den positiven Eigenschaften der SPH Formulierungen. Die attraktivsten Vorteile für die Siedlungswasserwirtschaft sind im numerischen Bereich die von vorn herein gegebene Massentreue, die Berechnung von Mehrphasenflüssigkeiten, frei schwimmende Körper und eine realistische Berechnung von Advektion und Diffusion [Aristodemo u. a. 2010; Gomez-Gesteira u. a. 2010; Colagrossi und Landrini 2003].

5.1.1 SPH Formulierung

Dadurch, dass SPH ein lagrangens Verfahren ist sind die Diskretisierungen der zugrunde liegenden Formeln sehr einfach gehalten. Sowohl die vollen Navier-Stokes Gleichungen als auch die von Saint-Venant entwickelten Flachwassergleichungen können mittels SPH berechnet werden [Vacondio u. a. 2010; Vacondio, Rogers, und Stansby 2011].

Die bereits erwähnte Glättungskurve **W** ist zentraler Bestandteil der SPH Formulierungen. Der Einfluss der Glättungskurve wird mit **h** beschränkt. Eine Feldfunktion **A** welches durch das Partikel **Ai** und der Position **r** beschrieben wird kann mittels folgender Gleichung ermittelt werden.

$$A_i = \int_{\Omega} A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}'$$

Entspricht die Glättungskurve dem Dirac'schen Delta, wenn **h** gegen null geht, so wird das Feld **A** korrekt repliziert, jedoch ist diese Variante nicht numerisch berechenbar. Eine numerische Approximation beinhaltet die Summierung der Volumen der benachbarten Partikel **j** und die Glättung der Feldfunktion A, wobei **r** und **rij** die Position des Partikels bzw. den Abstand zwischen zwei Partikel **i** und **j** beschreibt.

$$A_i = \sum_j A_j V_j W(\mathbf{r}_{ij},h)$$

$$V = \frac{m}{\rho}$$

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r}_{ij}, h)$$

Wird dieser Ansatz für die Kontinuitätsgleichung angewendet so erhält man folgende Gleichung für die Dichte.

$$\frac{d\rho_i}{dt} = \rho_i \sum_j \frac{m_j}{\rho_j} (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla W(\mathbf{r}_{ij}, h)$$

Dabei ist **m** die Masse, **rho** die Dichte und **v** die Geschwindigkeit des jeweiligen Partikels. Mittels einer Zustandsgleichung wird der Druck pro Partikel errechnet

$$p = B\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right)$$

Diese Zustandsgleichung beinhaltet eine Druckkonstante **B** und die Ruhedichte **rho0**. Die folgende Gleichung errechnet den Druckgradient und damit die Beschleunigung der Partikel.

$$\frac{dv_i}{dt} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij}\right) \nabla W(\mathbf{r}_{ij}, h)$$

Über die Beschleunigung wird dann mittels Integration die neue Position des Partikels berechnet.

$$\frac{dr_i}{dt} = v_i$$

5.1.2 Paralleliserung von SPH

Ein weiterer Vorteil von SPH ist, dass die Berechnungen pro Partikel unabhängig sind. Sprich, der Paralleliserungsgrad von SPH entspricht der Anzahl der Partikel welche je nach Diskretisierung in millionenfacher Höhe ist. Sieht man von der Nachbarschaftssuche ab, so ist SPH ein Verfahren welches prädestiniert für die Vielkernprozessoren, wie etwa GPUs ist. Es existieren bereits zahlreiche Implementierungen für GPUs welche auch das Problem der Nachbarschaftssuche gut in den Griff bekommen haben [Crespo u. a. 2011].

Der Aufwand von SPH ist ungleich höher als bei allen vorangegangenen Methoden, weshalb im Abstrakt die Kurve auch wieder nach oben zeigt. Jedoch wird der enorme Parallelisierungsgrad und die rasanten Entwick-

lungen bei den Vielkernprozessoren die Laufzeit von SPH soweit sinken lassen, dass Anwendungen im Massenmarkt immer attraktiver werden. Der Vorteil dieser Methode in der Siedlungswasserwirtschaft liegt dabei vor allem in einer sehr detaillierten Abbildung realer Prozesse, wodurch zum einen die Genauigkeit der Simulationsergebnisse erheblich gesteigert wird und zum anderen Berechnungen möglich werden, wie Sie derzeit kaum möglich sind (z.B. genaue Berechnung lokaler Fließverhältnisse in komplizierten Bauwerksgeometrien, Berechnung von Mehrphasenströmung für eine detaillierte Schmutzfrachtsimulation etc.).

Die Zukunft gehört den Vielkernprozessoren, jene Algorithmen die gut mit den Restriktionen der Vielkernprozessoren zurecht kommen werden in Zukunft bestehen und immer mehr an Verwendung finden. All diese Aspekte zwingen SPH eine rosige Zukunft auf.

Danksagung

Diese Arbeit wurde im Rahmen des Projektes PaCoWaDi vom Bundesministerium für Verkehr, Innovation und Technologie im Programm FIT-IT/ ModSim finanziell gefördert (FFG Projektnummer 2059687).

Literatur

Achleitner, S., M. Moderl, und W. Rauch. 2007. „CITY DRAIN\copyright-An open source approach for simulation of integrated urban drainage systems“. *Environmental modelling & software* 22 (8): 1184–1195.

Aristodemo, Francesco, Ivan Federico, Paolo Veltri, und Andrea Panizzo. 2010. „Two-phase SPH modelling of advective diffusion processes“. *Environmental Fluid Mechanics* 10 (4): 451–470.

Batchelor, G. K. 2000. *An Introduction to Fluid Dynamics*. Cambridge University Press.

Burger, G., S. Fach, H. Kinzel, und W. Rauch. 2010. „Parallel computing in conceptual sewer simulations“. *Water Science & Technology* 61 (2) (Januar): 283. doi:10.2166/wst.2010.798.

Colagrossi, Andrea, und Maurizio Landrini. 2003. „Numerical simulation of interfacial flows by smoothed particle hydrodynamics“. *Journal of Computational Physics* 191 (2): 448 – 475. doi:DOI: 10.1016/S0021-9991(03)00324-3.

Crespo, Alejandro C., Jose M. Dominguez, Anxo Barreiro, Moncho Gómez-Gesteira, und Benedict D. Rogers. 2011. „GPUs, a New Tool of Acceleration in CFD: Efficiency and Reliability on Smoothed Particle Hydrodynamics Methods“. *PLoS ONE* 6 (6): e20685. doi:10.1371/journal.pone.0020685.

Gingold, R. A., und J. J. Monaghan. 1977. „Smoothed particle hydrodynamics - Theory and application to non-spherical stars“. *Mon. Not. Roy. Astron. Soc.* 181 (November): 375–389.

Gomez-Gesteira, M., B.D. Rogers, R.A. Dalrymple, und A.J.C. Crespo. 2010. „State-of-the-art of classical SPH for free-surface flows“. *Journal of Hydraulic Research* 48: 6–27.

Grama, Ananth, Anshul Gupta, und George Karypis. 2003. *Introduction to Parallel Computing*. 0002 Aufl. Addison Wesley Pub Co Inc.

Huber, WC. 1995. „EPA Storm Water Management Model–SWMM“. *Computer Models of Watershed Hydrology* 1: 783–808.

Johansen, Niels Bent, und Danmarks Tekniske Højskole. Laboratoriet for Teknisk Hygiejne. 1985. „Discharge to Receiving Waters from Sewer Systems During Rain“. Dept. of Environmental Engineering, Technical University of Denmark.

Kuichling, E. 1889. „The relation between the rainfall and the discharge of sewers in populous districts“. *Transactions of the American Society of Civil Engineers* 20 (1): 1–56.

Lamb, Horace Sir 1849-1934. 2010. *Hydrodynamics*. Nabu Press.

Lucy, L. B. 1977. „A numerical approach to the testing of the fission hypothesis“. *Astronomical Journal* 82 (Dezember): 1013–1024.

Moore, G. E. 1998. „Cramming More Components Onto Integrated Circuits“. *Proceedings of the IEEE* 86 (1) (Januar): 82–85. doi:10.1109/JPROC.1998.658762.

Mulvaney, TJ. 1851. „On the use of self-registering rain and flood gauges in making observations of the relations of rainfall and flood discharges in a given catchment“. *Proceedings of the Institute of Civil Engineers of Ireland* 4: 18–31.

Olukotun, Kunle, Basem A. Nayfeh, Lance Hammond, Ken Wilson, und Kunyung Chang. 1996. „The case for a single-chip multiprocessor“. *SIGPLAN Not.* 31 (9) (September): 2–11. doi:10.1145/248209.237140.

Paulsen, O. 1987. „Kontinuierliche Simulation von Abflüssen und Stofffrachten in der Trennentwässerung:(Zusammenfassung, Summary)“.

Pham, D., S. Asano, M. Bolliger, M.N. Day, H.P. Hofstee, C. Johns, J. Kahle, u. a. 2005. „The design and implementation of a first-generation CELL processor“. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, 184–592.

Rauch, W., und H. Kinzel. 2007. *KAREN-User Manual." hydro-IT GmbH*. Innsbruck.

Rauch, W., V. Krejci, und W. Gujer. 2002. „REBEKA–a software tool for planning urban drainage on the basis of predicted impacts on receiving waters“. *Urban water* 4 (4): 355–361.

Seiler, Larry, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, u. a. 2008. „Larrabee: a many-core x86 architecture for visual computing“. *ACM Trans. Graph.* 27 (3) (August): 18:1–18:15. doi:10.1145/1360612.1360617.

Sutter, H. 2011. „Welcome to the Jungle“. *Sutter's Mill*. http://herbsutter.com/2011/12/29/welcome-to-the-jungle/.

Sutter, Herb. 2005. „The free lunch is over: A fundamental turn toward concurrency in software“. *Dr. Dobb's Journal* 30 (3): 202–210.

Vacondio, R., P. Mignosa, BD Rogers, und PK Stansby. 2010. „2-D numerical modeling of rapidly varying shallow water flows by Smoothed Particle Hydrodynamics technique“. *Proc. of Riverflow*.

Vacondio, R., B. D. Rogers, und P. K. Stansby. 2011. „Smoothed Particle Hydrodynamics: Approximate zero-consistent 2-D boundary conditions and still shallow-water tests“. *International Journal for Numerical Methods in Fluids*: n/a–n/a. doi:10.1002/fld.2559.

Hinweis

Teile dieses Textes (Kapitel 1 bis 3) wurden in ähnlicher Form bereits 2010 in der Zeitschrift: Österreichische Wasser- und Abfallwirtschaft, 62(3-4), 43-50 abgedruckt.

Korrespondenz an:

Gregor Burger
Technikerstraße 13
A-6020 Innsbruck
Tel: +43 512 507 6988
Fax: +43 512 507 2911
Email: gregor.burger@uibk.ac.at