Parallele Lösung eines stetigen deterministischen Steuerproblems

Thomas Pohl, Holgar Blaar, Wilfried Grecksch

Martin-Luther-Universität Halle-Wittenberg
Fachbereich Mathematik und Informatik
Institut für Stochastik und Optimierung
Institut für Informatik

email: pohl@infpar.informatik.uni-halle.de, blaar@informatik.uni-halle.de,
grecksch@mathematik.uni-halle.d400.de

Zusammenfassung

Zur parallelen Lösung des allgemeinen Steuerproblems werden drei prinzipielle Ansätze untersucht: die Lösung über das adjungierte System, der Einsatz eines stochastischen Gradientenverfahrens sowie die Verwendung eines stochastischen Suchverfahrens. Die strengen sequentiellen Abhängigkeiten in den Algorithmen konnten aufgebrochen und einer parallelen Lösung zugänglich gemacht werden.

1. Einführung

Es wird die Parallelisierung von Algorithmen untersucht, die folgendes allgemeines Steuerproblem lösen:

$$J(x,u) = \int_0^T g(x(t), u(t), t) \overset{!}{=} \min$$

mit $\dot{x}(t) = f(x(t), u(t), t),\ x(0) = x_0$.

Dabei sind $u : [0,T] \to U \subseteq \Re^m$ die Steuerfunktion und $x : [0,T] \to X \subseteq \Re^n$ die Zustandsfunktion. Da diese Probleme in der Praxis oft rechenintensiv sind, ist der Einsatz von parallelen Algorithmen erwägenswert. Wegen starker sequentieller Abhängigkeiten ist die Parallelisierung schwierig. Wright löste in [1] derartige Steuerprobleme, indem er sogenannte "Separator-Variablen" einführte und über die Stufen parallelisierte. Dort bestand die Schwierigkeit darin, die Riccati-Gleichungen zu lösen. Da diese numerisch oft schwierig zu lösen sind bzw. hohen Rechenaufwand erfordern und oft an den linearen Regulator gebunden sind, ist die Betrachtung weiterer Lösungsmöglichkeiten von Interesse. In den vorliegenden Untersuchungen wird das Steuerproblem mit Hilfe von stochastischen Verfahren bearbeitet. Die numerischen Experimente wurden auf einem Transputersystem mit $p=8$ Prozessoren vorgenommen (verteilter Speicher), die Implementierung erfolgte in der Sprache Occam. Sämtliche Algorithmen werden an einem einfachen Beispiel demonstriert:

$$J(x,u) = \int_0^1 [6u^2(t) + 2x_1^2(t) + x_2^2(t)]dt \overset{!}{=} \min$$

mit $\dot{x}_1(t) = x_2(t),\ x_1(0) = 0$
$\dot{x}_2(t) = -x_1(t) + u(t),\ x_2(0) = 0$
$|u(t)| \le 1.$

Für solche Systeme wurden in [1] parallele deterministische Algorithmen getestet.

2. Lösung des Steuerproblems über das adjungierte System

Für das zeitdiskrete System

$$x^j_{i+1,1} := x^j_{i,1} + hx^j_{i,2},\ x^j_{1,1} = 0 \qquad ,i=1,\ldots,N$$

$$x^j_{i+1,2} := x^j_{i,2} - hx^j_{i,1} + hu_i,\ x^j_{1,2} = 0 \qquad ,i=1,\ldots,N$$

$$J_N := \sum_{i=1}^{N} \left(6\left(u^j_i\right)^2 + 2\left(x^j_{i,1}\right)^2 + \left(x^j_{i,2}\right)^2 \right) h$$

ergeben sich die adjungierten Gleichungen zu

$$p^j_{i,1} := \frac{h\left(-p^j_{i+1,2} + 2hx^j_{i,2} - 4x^j_{i,1}\right) + p^j_{i+1,1}}{h^2+1},\ p^j_{N+1,1} = 0 \qquad ,i=N,\ldots,1$$

$$p^j_{i,2} := \frac{h\left(p^j_{i+1,1} - 4hx^j_{i,1} - 2x^j_{i,2}\right) + p^j_{i+1,2}}{h^2+1},\ p^j_{N+1,2} = 0 \qquad ,i=N,\ldots,1$$

wobei $h=1/N$, i der Stufenindex und j der Iterationsindex ist. Der Gradient des Funktionals ergibt sich zu $\frac{\partial H}{\partial u}$, wobei H die in [2] eingeführte Hamiltonfunktion ist. Die Anwendung des Gradientenverfahrens ergibt

$$u^{j+1}_i := u^j_i + \lambda \frac{\partial H\left(x^j_i, u^j_i, p^j_i\right)}{\partial u},\ u = u^j_i \qquad ,i=1,\ldots,N \quad .$$

Der sequentielle Algorithmus besteht nun aus folgenden Schritten:

S1 $u_i := u \in U\ \forall i \in [1,N];\ x_1 := x(0);\ p_{N+1} := 0$
$\quad$ *While eps* $> \varepsilon$ *do*
$\quad$ *begin*
S2 $\quad$ *FOR* $i := 1$ *to* N *do* $x_{i+1} := f(x_i, u_i)h + x_i$
S3 $\quad$ *FOR* $i := N$ *to* 1 *do* $p_i := \tilde{g}\,(x_i, p_{i+1}, u_i)$
S4 $\quad$ *FOR* $i := 1$ *to* N *do* $u_i := u_i + \lambda \frac{\partial H(x_i, u_i, p_i)}{\partial u},\ u = u_i$
S5 $\quad$ *Berechnung von* $J^j(x^j, u^j)$
S6 $\quad$ $eps := \left| J^j(x^j, u^j) - J^{j-1}(x^{j-1}, u^{j-1}) \right|$
$\quad$ *end*

Hierbei bezeichnet $\tilde{g}$ die Auflösung der adjungierten Gleichungen nach p_{i+1}. Die Parallelisierung dieses Algorithmus geschieht in der Weise, daß man die N Stufen in p Teile teilt, sodaß jeder Prozessor eine gleich große Anzahl von Stufen bearbeitet. Bei genauerer Betrachtung der Schritte *S2* und *S3* des sequentiellen Algorithmus erkennt man, daß sie nur sequentiell ausführbar sind, da man die x-Werte zuerst in aufsteigender Folge berechnen muß, um anschließend die p-Werte in absteigender Folge berechnen zu können. Aus diesem Grunde wurde eine Pipelinestruktur gewählt. Da die Parallelisierung auf einem Transputersystem erfolgt, also einem Parallelrechner mit verteiltem Speicher, müssen noch zusätzlich Datentransporte zwischen den Prozessoren stattfinden. Das hat zur Folge, daß die Bearbeitung der Schritte *S2* und *S3* auf solch einem Parallelrechner mehr Zeit in Anspruch nimmt als bei einem sequentiellen Rechner. Indessen können die Schritte *S4* und *S5* "sehr gut" parallelisiert werden, da jeder Prozessor für "seine" Stufen unabhängig von den anderen die neuen Werte der Steuerfunktion und die Teilsumme $J^j_p(x,u)$ berechnen kann. Am Ende einer solchen Iteration wird dann $J^j(x,u)$ berechnet und mit dem Wert $J^{j-1}(x,u)$ verglichen. Der Speedup bei dieser Art von Parallelisierung hängt im wesentlichen von dem Verhältnis der Rechenaufwände der Schritte *S2* und *S3*, die "nicht" parallelisiert werden können, und den Schritten *S4* und *S5* ab. Das obige Beispiel wurde mit 8 Prozessoren parallelisiert, wobei ein Speedup von nur 1,01 auftrat, was an dem ungünstigen Verhältnis der

Rechenaufwände und der Zeiten für die Datentransporte liegt. Aus diesem Grund wurde versucht, die Zeiten für die Datentransporte und die Wartezeiten zu vermindern. Um dieses zu erreichen, wurde nur noch bei der ersten und dann erst wieder nach k Iterationen ein Datentransport durchgeführt (bei den Berechnungen der Schritte *S2* und *S3*). Das bedeutet, daß beim Prozessor *(j+1)* der $x_{j\frac{N}{p}-1}$ - und $p_{\frac{j}{p}N+1}$ -Wert für *(k-1)* Iterationen festgehalten wird. Durch dieses Verfahren fallen zwar die Transport- und Wartezeiten weg, jedoch ändert sich das Konvergenzverhalten, da die Prozessoren nur nach *k* Iterationen ihre Werte aktualisieren. Aus diesem Grund darf man k nicht all zu groß wählen, da sonst der Vorteil durch die Reduzierung der Transport- und Wartezeiten wegen erhöhter Iterationszahl aufgehoben wird oder das Verfahren sogar divergiert. An dem Beispiel wurden folgende Werte gemessen:

ε	k	ZEIT in Sekunden sequentiell	parallel	Speedup (p=8)	Effizienz
0,1	1	1,3691	1,3530	1,01	0,13
0,1	2		0,8218	1,67	0,21
0,1	5		0,5121	2,68	0,33
0,01	1	2,1137	2,0914	1,01	0,13
0,01	2		1,2653	1,67	0,21
0,01	5		0,7795	2,71	0,34
0,01	10		0,6496	3,25	0,41
0,001	1	2,7837	2,7562	1,01	0,13
0,001	2		1,7087	1,63	0,20
0,001	5		1,0468	2,66	0,33
0,001	10		0,8582	3,24	0,40
1,0E-6	1	4,9430	4,8980	1,01	0,13
1,0E-6	2		3,0079	1,64	0,21
1,0E-6	5		1,8488	2,67	0,33
1,0E-6	10		1,5416	3,20	0,40
1,0E-15	1	11,4946	11,3970	1,01	0,13
1,0E-15	2		6,9104	1,66	0,21
1,0E-15	5		4,2550	2,70	0,34
1,0E-15	10		3,4795	3,30	0,41
1,0E-30	1	22,4379	22,2537	1,01	0,13
1,0E-30	2		13,4737	1,67	0,21
1,0E-30	5		8,2804	2,71	0,34
1,0E-30	10		6,7730	3,31	0,41
1,0E-100	1	73,5047	72,9177	1,01	0,13
1,0E-100	2		44,0724	1,67	0,21
1,0E-100	5		27,0994	2,71	0,34
1,0E-100	10		22,0648	3,33	0,42

Da diese Art der Lösung von Steuerproblemen (ohne Datenransportverzögerung) sich nicht besonders gut parallelisieren läßt (gute Effizienz nur bei Problemen mit hohem Rechenaufwand der Hamiltonfunktion), sind Methoden sinnvoll, die die Berechnung des Gradienten unter Nutzung des adjungierten Systemes umgehen. Zusätzlich werden noch Aspekte der stochastischen Optimierung einbezogen.

3. Lösung durch ein stochastisches Gradientenverfahren

Durch Anwendung der stochastischen Quasigradientenmethode ergibt sich folgender sequentieller Algorithmus:

S1 $\tilde{u}_i := u_i := u \in U \; \forall i \in [1,N];\; x_1 := x(0);\; \tilde{x}_1 := x(0)$
$FOR\, j := 1 \; to \; n-1 \; do$
$begin$
S2 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$
S3 $Berechnung \; von \; J(x,u)$
S4 $FOR\; i := 1 \; to \; p-1 \; do$
$begin$
$k := random[N]$
$\tilde{u}_k := u_k + \delta$
$FOR\; l := 1 \; to \; N \; do \; \tilde{x}_{l+1} := f(\tilde{x}_l, \tilde{u}_l)h + \tilde{x}_l$
$Berechnung \; von \; \tilde{J}(\tilde{x}, \tilde{u})$
$u_k := u_k - \alpha \frac{J(x,u) - \tilde{J}(\tilde{x},\tilde{u})}{\delta}$
end
end
$\tilde{u}_k := u_k$ (für alle k, die gewählt wurden)
S5 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$
S6 $Berechnung \; von \; J(x,u)$

Hierbei sind n die Anzahl der Iterationen, α die Schrittweite und δ eine genügend kleine positive Zahl. Bei diesem Algorithmus wurden wegen der besseren Vergleichsmöglichkeit zum parallelen Algorithmus eine feste Iterationszahl als Abbruchkriterium und beim Schritt *S4* *(p-1)* Iterationen gewählt.

Beim parallelen Algorithmus wird eine Ringtopologie verwendet. Ein Masterprozessor dient zur Berechnung von *J(x,u)*. Die anderen Prozessoren wählen jeweils aus ihrem Teilbereich der Stufen einen Index k, wobei die Aufteilung wie beim vorigen Agorithmus ist, und berechnen dann die $\tilde{x}$ *–Werte* und $\tilde{J}(\tilde{x}, \tilde{u})$. Anschließend sendet der Masterprozessor sein *J(x,u)* durch den Ring zu allen Prozessoren. Danach berechnen die anderen Prozessoren den Differenzenquotienten und den neuen u_i-Wert, der im Ring an jeden Prozessor geschickt wird. Dieser Prozeß wiederholt sich *(n-2)* mal.

Paralleler Algorithmus :

Masterprozessor:

S1 $\tilde{u}_i := u_i := u \in U \; \forall i \in [1,N];\; x_1 := x(0);\; \tilde{x}_1 := x(0)$
$FOR\, j := 1 \; to \; n-1 \; do$
$begin$
S2 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i;$ *Empfang der u – Werte*
S3 $Berechnung \; von \; J(x,u)$ *; Senden von J(x,u) an die anderen Prozessoren*
end
S5 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$
S6 $Berechnung \; von \; J(x,u)$

Prozessoren 2,...,p:

$$
\begin{aligned}
&S1\quad \tilde{u}_i := u_i := u \in U\ \forall i \in [1,N];\ x_1 := x(0);\ \tilde{x}_1 := x(0)\\
&\quad FOR\ j := 1\ to\ n-1\ do\\
&S4\quad begin\\
&\qquad k := random[N]\\
&\qquad \tilde{u}_k := u_k + \delta\\
&\qquad FOR\ l := 1\ to\ N\ do\ \tilde{x}_{l+1} := f(\tilde{x}_l, \tilde{u}_l)h + \tilde{x}_l\\
&\qquad Berechnung\ von\ \tilde{J}(\tilde{x},\tilde{u});\ Empfang\ von\ J(x,u)\\
&\qquad u_k := u_k - \alpha\frac{J(x,u)-\tilde{J}(\tilde{x},\tilde{u})}{\delta}\quad ;Empfang\ und\ Weiterleiten\ der\ u\text{-}Werte\\
&\quad end
\end{aligned}
$$

Da hier jeder Prozessor bis auf den dazu relativ geringen Datentransportaufwand unabhängig rechnen kann, ist ein "guter" Speedup erzielbar, der mit wachsendem n größer wird. Das liegt daran, daß der Masterprozessor am Ende noch den aktuellen Wert von J(x,u) berechnen muß. Die anderen Prozessoren sind in dieser Zeit nicht ausgelastet. Dieser Anteil verringert sich mit zunehmendem Gesamtrechenaufwand (wachsendes n). Dieses soll an folgender Überlegung verdeutlichst werden:

Es sei a = Rechenaufwand für $u_i := u_i - \alpha\frac{J(x,u)-\tilde{J}(x,u)}{\delta}$

b = Rechenaufwand für $x_{i+1} := f(x_i, u_i)h + x_i$

c = Rechenaufwand für $J(x,u)$ bzw. $\tilde{J}(x,u)$.

Dann ergeben sich folgende Kosten für den sequentiellen bzw. parallelen Algorithmus:

$$
\begin{aligned}
T_1 &= (n-1)[(p-1)a + p(b+c)] + b + c\\
T_p &= (n-1)(a+b+c) + b + c
\end{aligned}
$$

und damit ein Speedup

$$S_p = \frac{(n-1)[(p-1)a+p(b+c)]+b+c}{(n-1)(a+b+c)+b+c}.$$

Da der Rechenaufwand a relativ klein zu denen von b und c ist, gilt für den Grenzwert:

$$\lim_{n\to\infty} S_p = p \quad \text{(ohne Berücksichtigung der Datentransportzeiten).}$$

Die Meßwerte für das einfache Beispiel verdeutlichen, daß der Speedup mit zunehmendem n wächst. Da hier noch Zeiten für die Datentransporte anfallen, ist der Speedup kleiner als p (hier 8).

n	ZEIT in Sekunden sequentiell	parallel	Speedup (p=8)	Effizienz
1	0,3194	0,1189	2,69	0,34
10	2,8260	0,6242	4,53	0,57
100	27,8918	5,6786	4,91	0,61
1000	278,5500	56,2234	4,95	0,62

Dieses Verfahren bietet sich auch deshalb an, weil es bei komplexen Aufgaben (*u* mehrdimensional, Zielfunktion rechenintensiver) schwierig bzw. mit sehr hohem Aufwand verbunden ist, den Gradienten exakt zu bestimmen.

4. Lösung mittels eines stochastischen Suchverfahrens

Die Implementierung auf dem Parallelrechner ist analog der beim stochastischen Gradientenverfahren. Der Grundgedanke des sequentiellen Algorithmus ist es, von den Stufen 1,...,N eine auszuwählen, um anschließend eine Zufallsgröße Y zu bilden, nach der entschieden wird, in welcher Richtung die Steuerung u_i verändert werden soll. Danach wird getestet, ob die Richtung eine Verbesserung der Zielfunktion war oder nicht (Veränderung wird rückgängig gemacht). Bei der Durchführung wurden aus dem Grunde nur einzelne Stufen ausgewählt, da die unteren Stufen (wegen der Rekursion) einen größeren Einfluß auf die Entwicklung des Systems haben als obere, was bewirken würde, daß man "richtige" Richtungen der oberen Stufen bei "falschen" Richtungen der unteren Stufen mit verneinen würde.

Sequentieller Algorithmus:

S1 $\tilde{u}_i := u_i := u \in U \; \forall \, i \in [1, N]; \; x_1 := \tilde{x}_1 := x(0)$

$\quad FOR\, j := 1 \; to \; n-1 \; do$

$\quad begin$

S2 $\quad FOR\, i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$

S3 $\quad Berechnung \; von \; J(x, u)$

S4 $\quad FOR\, l := 1 \; to \; p-1 \; do$

$\qquad begin$

$\qquad\quad k := random[N]$

$\qquad\quad \tilde{k} := random$

$\qquad\quad IF\; \tilde{k} \leq 0,5 \; then \; \tilde{u}_k := \tilde{u}_k - \delta$

$\qquad\qquad\qquad else \; \tilde{u}_k := \tilde{u}_k + \delta$

$\qquad\quad FOR\, l := 1 \; to \; N \; do \; \tilde{x}_{l+1} := f(\tilde{x}_l, \tilde{u}_l)h + \tilde{x}_l$

$\qquad\quad Berechnung \; von \; \tilde{J}(\tilde{x}, \tilde{u})$

$\qquad\quad IF\; \tilde{J}(\tilde{x}, \tilde{u}) > J(x, u) \; then \; \tilde{u}_k := u_k$

$\qquad\qquad\qquad\qquad else \; u_k := \tilde{u}_k$

$\qquad end$

$\quad end$

S5 $FOR\, i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$

S6 $Berechnung \; von \; J(x, u)$

Im parallelen Algorithmus dient wieder ein Masterprozessor zur Bestimmung der x-Werte und von $J(x,u)$. Die anderen berechnen wie beim obigen Verfahren die $\tilde{x}-Werte$ und $\tilde{J}(\tilde{x}, \tilde{u})$.

Paralleler Algorithmus:

Masterprozessor:

S1 $\tilde{u}_i := u_i := u \in U \; \forall i \in [1,N];\; x_1 := x(0);\; \tilde{x}_1 := x(0)$

$FOR\, j := 1 \; to \; n-1 \; do$

$begin$

S2 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$

S3 $Berechnung \; von \; J(x,u)$

end

S5 $FOR\; i := 1 \; to \; N \; do \; x_{i+1} := f(x_i, u_i)h + x_i$

S6 $Berechnung \; von \; J(x,u)$

Prozessoren 2,...,p:

S1 $\tilde{u}_i := u_i := u \in U \; \forall i \in [1,N];\; x_1 := x(0);\; \tilde{x}_1 := x(0)$

$FOR\, j := 1 \; to \; n-1 \; do$

S4 $begin$

$k := random[N]$

$\tilde{k} := random$

$IF\; \tilde{k} \leq 0,5 \; then \; \tilde{u}_k := \tilde{u}_k - \delta$

$else \; \tilde{u}_k := \tilde{u}_k + \delta$

$FOR\; l := 1 \; to \; N \; do \; x_{l+1} := f(x_l, u_l)h + x_l$

$Berechnung \; von \; \tilde{J}(\tilde{x}, \tilde{u})$

$IF\; \tilde{J}(\tilde{x}, \tilde{u}) > J(x,u) \; then \; \tilde{u}_k := u_k$

$else \; u_k := \tilde{u}_k$

end

Für das Beispiel wurden folgende Werte erzielt:

n	ZEIT in Sekunden sequentiell	ZEIT in Sekunden parallel	Speedup (p=8)	Effizienz
1	0,2131	0,0805	2,65	0,33
10	1,8857	0,4281	4,40	0,55
100	18,6111	3,9038	4,77	0,60
1000	185,8685	38,6586	4,81	0,60

Auch hier gilt, daß der Speedup mit zunehmendem n wächst. Dieses Verfahren bietet sich bei sehr komplexen Aufgaben an, da hier kein Gradient gebildet werden muß. Es werden Richtungen ausgesucht und dann geprüft, ob sie eine Verbesserung (Minimum, Maximum) der Zielfunktion liefern.

Eine andere Möglichkeit der Parallelisierung besteht darin, verschiedene Startpunkte u_i zu wählen und von diesen die stochastische Suche zu beginnen. Eine Einschätzung des Speedups bzw. der Effizienz ist allerdings schwierig, weil er von den gewählten Punkten des parallelen bzw. sequentiellen Algorithmus abhängig ist.

5. Zusammenfassung und Ausblick

Die vorgestellten Ergebnisse zeigen, daß die strengen sequentiellen Abhängigkeiten bei der Lösung des allgemeinen Steuerproblems mit gezielten Parallelisierungen aufgebrochen werden können und so diese Problemklasse parallelen Berechnungen zugänglich wird. Weitere Bearbeitungsrichtungen können sein, andere Topologien (z. B. Stern) zu untersuchen, um den Kommunikationsoverhead gezielt zu verringern. Dabei ist allerdings zu beachten, daß eine für den jeweiligen parallelen Algorithmus optimale virtuelle Topologie nicht auf jedem Rechner mit entsprechender Effizienz realisierbar sein wird. Bisher sind auch theoretische Aussagen zum Konvergenzverhalten des Algorithmus aus Abschnitt 2 in Abhängigkeit von dem die Iteration beeinflussenden Parameter k offen und bedürfen tiefergehender Untersuchungen.

Literatur

[1] Wright, Stephen J: Partitioned dynamic programming for optimal control, SIAM J. Optim. 1, No. 4, 1991, 620 -642

[2] Fleming,Wendell H., Rishel, Raymond W.: Deterministic and Stochastic Optimal Control, Springer, 1975