# افزایش همزمانی در فراخوانی‌های غیرهمزمان با جابجایی دستورالعملها

سعید پارسا*، روزبه معانی†

## چکیده

**استفاده از فراخوانی های غیرهمزمان یکی از روشهای مرسوم برای ایجاد همروندی جهت تسریع اجرای برنامه های توزیع شده می باشد. اما این همروندی اغلب تا حداکثر مقدار ممکن ایجاد نمی‌گردد. در واقع، وابستگی دستورات بعد از فراخوانی های غیر همزمان به مقادیر تاثیر پذیر از فراخوانیها، موجب کاهش همروندی در اجرای همزمان دو تابع فراخواننده و فراخوانده شده می شود. در این مقاله نشان داده شده که برای افزایش میزان همروندی می توان دستورات را به گونه ای جابجا کرد که دستوراتی که از مقادیر توابع غیر همزمان استفاده می کنند تا حد امکان از محل فراخوانی تابع دور شوند. بدین ترتیب همانطور که در عمل نشان داده ایم، این جابجایی دستورالعملها موجب افزایش قابل توجهی در میزان همروندی و بالنتیجه سرعت اجرای برنامه شده است.**

## کلمات کلیدی

**فراخوانی های غیر همزمان، افزایش همروندی، جابجایی دستورات**

# Increasing concurrency in asynchronous calls with instruction reordering

Saeed Parsa, Rouzbeh Maani

## Abstract

Using asynchronous calls is a common approach for creation of concurrency in distributed programs. But sometimes this concurrency cannot reach to its maximum amount. In fact, dependencies of the instructions which are executed after asynchronous calls to the values which are affected by the calls decrease the amount of concurrency between the caller and the called methods. In this paper it is shown that to increase the amount of concurrency it is possible to reorder the instructions in such ways that instructions which use the results of an asynchronous method locate as far as possible from the position of the method call. Therefore as shown in practice, this instruction reordering causes a noticeable increase in concurrency and consequently speeding up the program.

## Keywords

Asynchronous Calls, increasing concurrency, instruction reordering.

* دانشیار و عضو هیأت علمی دانشگاه، دانشگاه علم و صنعت ایران، دانشکده کامپیوتر،parsa@iust.ac.ir.

† دانشجوی کارشناسی ارشد نرم افزار، دانشگاه علم و صنعت ایران، دانشکده کامپیوتر، rouzbeh_maani@yahoo.com.

## ۱- مقدمه

سرعت بالای اجرایی در دنیای صنعتی و بخصوص در اجرای برنامه های علمی بسیار حایز اهمیت می باشد در این راستا جهت افزایش میزان توازی در اجرای برنامه ها تلاشهای بسیار زیادی انجام گرفته است. هدف در این مقاله افزایش میزان همروندی حاصل از اجرای فراخوانیهای غیر همگام تا حد ممکن در اجرای برنامه های توزیعی می باشد.

روش جابجایی و یا زمان بندی دستورات[1] یکی از راه حلهایی است که برای افزایش توازی در کد برنامه ها مورد استفاده قرار میگیرد[2]. این روش و الگوریتمهای ارائه شونده در آن، به معماری سیستمهای مورد استفاده وابسته است و اهداف مختلفی را دنبال می کند. از جمله این اهداف می‌توان به افزایش خروجی[3] ، مینیمم کردن فشار ثباتها [4]، کاهش اثر نبود اطلاعات در کاشه و ترکیبی از این اهداف اشاره نمود[5].

بطور کلی استفاده از فراخوانی های راه دور غیر همزمان یکی از روشهای ایجاد توازی در برنامه های توزیع شده محسوب می‌شود که بویژه در تبدیل کد برنامه های سریال به موازی کاربرد فراوانی دارد[6]. در این مقاله ایده جابجایی دستورات برای افزایش میزان توازی در فراخوانی های غیر همزمان مورد استفاده قرار گرفته است. این الگوریتم با توجه استفاده از فراخوانی های راه دور در ایجاد توازی ارائه شده است. در ادامه در بخش ۲ مشکل اصلی در کاهش همروندی و محدودیتهایی که میبایست برای جابجایی دستورات در نظر گرفته شود آمده است. در بخش ۳ الگوریتم پیشنهادی ارائه شده است. بخش ۴ شامل نتایج بدست آمده از اعمال الگوریتم بر روی یک کد توزیع شده است و در بخش ۵ نتیجه گیری ارائه گردیده است.

## ۲- الگوریتم جابجایی دستورات

در فراخوانی‌های راه دور، بهترین حالت زمانی است که نتایج یک فراخوانی راه دور پس از زمان اتمام فراخوانی مورد استفاده قرار گیرند. الگوریتم پیشنهادی سعی می‌کند که فاصله بین فراخوانی های راه دور و دستورات استفاده کننده از آنها را زیاد کند. برای اجرای الگوریتم لازم است تا زمان تخمینی اجرای هر دستورالعمل مشخص باشد. برای این تخمین می‌توان از الگوریتم های مختلفی استفاده نمود[7].

یک دستورالعمل زمانی می‌تواند جابجا شود که هیچگونه تداخلی در وابستگی داده ای و وابستگی کنترلی بوجود نیاورد[1]. در بخشهای بعدی این وابستگی ها و راه حل حفظ آنها در الگوریتم پیشنهادی بررسی شده است.

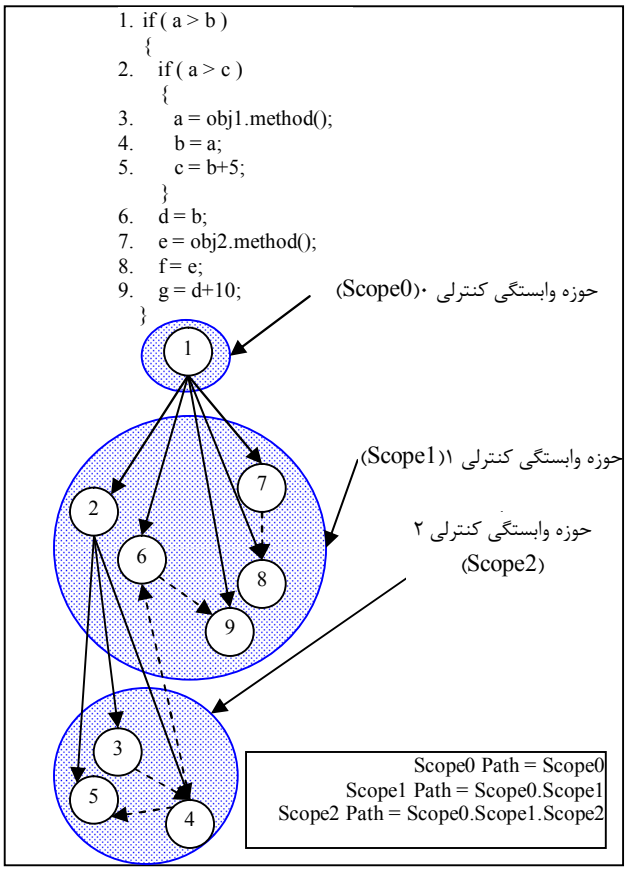
### ۲-۱- وابستگی کنترلی

در تعریف یک دستور به دستور دیگر وابستگی کنترلی دارد اگر اجرای دستور وابسته به نتیجه اجرای آن دستور باشد. برای جلوگیری از تداخل در وابستگی کنترلی، در الگوریتم پیشنهادی دستورات در حوزه های وابستگی کنترلی دسته بندی می‌شوند و جابجایی تنها بین دستوراتی صورت می‌گیرد که در یک حوزه وابستگی کنترلی قرار دارند. در شکل ۱ یک کد نمونه به همراه گراف وابستگی کنترلی که در آن حوزه های وابستگی کنترلی مشخص شده است نمایش داده شده است. در این گراف وابستگی های کنترلی با فلش های عادی نشان داده شده است.

### ۲-۲- وابستگی داده ای

بطور کلی سه نوع وابستگی داده ای بین دستورات وجود دارد: خواندن بعد نوشتن، نوشتن بعد خواندن و نوشتن بعد نوشتن[1]. می‌توان وابستگی های داده ای را نیز در قالب یک گراف نمایش داد که در آن گره ها نماینده دستورات و لبه ها نماینده وابستگی‌هاست. در شکل ۱ گراف وظایف (گرافی که شامل هر دو نوع وابستگی داده ای و کنترلی است) در یک قطعه کد نمونه نشان داده شده است. وابستگی داده‌ای در این گراف با خط چین مشخص شده است.

برای حفظ وابستگی‌های داده ای در الگوریتم از متدی به نام ResolveDependency استفاده می شود. این متد برای یک دستورالعمل کنترل می‌کند که تمامی دستوراتی که در گراف وابستگی داده‌ای به عنوان پدر آن دستورالعمل هستند قبل از آن دستور دیده شوند.



**شکل (۱): گراف وابستگی کنترلی و داده ای(گراف وظایف)**

## ۳- توضیح الگوریتم

همانطور که اشاره شد هدف الگوریتم دور کردن دستورات استفاده کننده از نتایج متدهای راه دور از دستورات فراخوانی آن متد هاست تا زمان اجرای همزمان دو متد فراخواننده و فراخوانده شده به حداکثر ممکن برسد. در این الگوریتم دستورات به ۳ دسته کلی تقسیم می‌شوند:

- ***دستورات فراخوانی (CallInstruction):*** دستوراتی که در آنها یک متد راه دور فراخوانی می‌شود.
- ***دستورات استفاده (UseInstruction):*** دستوراتی که از نتایج فراخوانی های راه دور استفاده می‌کنند.
- ***دستوراتعادی(CommonInstruction):***سایر دستورات

بطور کلی می‌توان الگوریتم را شامل ۴ مرحله دانست. در مرحله اول سعی می‌شود تا فراخوانی های راه دور به بالاترین مکان ممکن در داخل کد برنامه انتقال داده شوند. اولویت اصلی در انتقال فراخوانی ها زمان تکمیل اجرای آنهاست. برای این منظور تمامی گره‌هایی که در گراف وابستگی کنترلی و داده‌ای (گراف وظایف) قبل از گره مربوط به فراخوانی مورد نظر قرار دارند انتخاب می‌شوند. حال در میان این گره‌ها باز هم دستورالعمل فراخوانی با بیشترین زمان اجرا پیدا نموده و این عمل را بطور خود بازگشتی آنقدر تکرار می‌شود تا اینکه فراخوانی راه دوری باقی نماند. پس از آن تمامی دستورات عادی و سپس دستورات استفاده کننده از نتایج متدهای راه دور انتخاب می‌شوند.

هر زمان که یک دستور انتخاب شود میبایست ابتدا دستورات پدر آن در گراف وابستگی کنترلی انتخاب شوند و سپس دستور فراخوانی مربوطه انتخاب شود. پس از انتخاب هر دستور با استفاده از متد ResolveDependency وابستگی‌های داده‌ای حفظ می‌شود.

در مرحله دوم دستوراتی که از نتایج متدها استفاده می‌کنند و هنوز انتخاب نشده اند انتخاب می‌شوند.

در مرحله سوم در میان کل دستورات بزرگترین زمان توقف پیدا می‌شود. پس از بدست آمدن بزرگترین توقف تمامی مکانهایی که بین فراخوانی تابع تا استفاده از نتایج آن وجود دارد بدست آمده و می‌توان این مکانها را به عنوان محل‌های کاندید درج باقی دستورات، مورد استفاده قرار داد. بدین ترتیب در این مرحله سعی می‌شود تا در صورت امکان دستورات باقیمانده در محل‌های توقف درج شوند تا زمان‌های توقف کاهش یابد.

مرحله چهارم در واقع پس از انتخاب تمامی دستورات انجام می‌گیرد و سعی می‌گردد تا اگر امکان کاهش زمان توقف با جابجایی دستورات موجود در حوزه های وابستگی کنترلی وجود دارد، دستورات جابجا شوند. برای این منظور در هر حوزه بزرگترین توقف پیدا می‌شود. تمامی دستورات بعد از توقف مورد آزمایش قرار می‌گیرند که آیا می‌توانند به قبل از دستور توقف کننده منتقل شوند و در صورت امکان به قبل از آن منتقل می‌گردند. همین امر برای دستورات قبل از فراخوانی نیز صورت می‌گیرد.

## ۴- نتایج اعمال الگوریتم

برای بررسی و برآورد تاثیر الگوریتم بر روی برنامه ها، الگوریتم پیشنهادی بر روی برنامه فروشنده دوره گرد اعمال شد. این برنامه دارای ۱۷ کلاس و ۱۲۹ متد است. زمان همروندی پیش و پس از اعمال الگوریتم بر روی برنامه مورد نظر به ازای ورودی های مختلف در شکل ۴ نمایش داده شده است. در شکل ۵ میزان افزایش همروندی بعد از اعمال الگوریتم نسبت به حالت قبل از آن نشان داده شده است.

## ۵- نتیجه

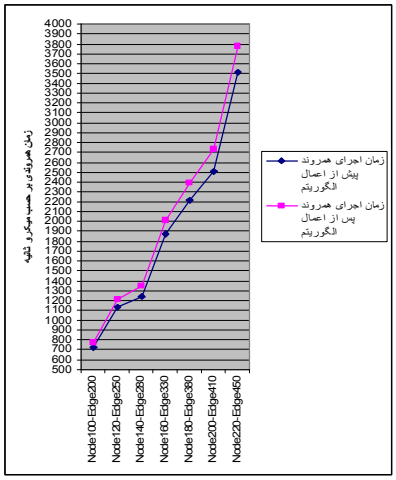
در این مقاله، الگوریتمی نوین برای افزایش میزان همروندی در اجرای فراخوانیهای غیر همزمان در برنامه های توزیعی ارایه شد. برای این منظور در درجه اول جمله های فراخوانی به بالاترین مکان ممکن انتقال داده می شوند. در هنگام انتقال الویت به فراخوانی متدهای بزرگتر باید داده شود. به همین ترتیب جملات استفاده کننده از نتایج فراخوانیها باید به پایینترین مکان ممکن انتقال داده شوند. در اینجا اولویت به مکان استفاده با طولانی ترین زمان توقف باید داده شود. در مرحله چهارم الگوریتم سعی می‌شود تا در صورت امکان جابجایی زمان توازی به بالاترین حد خود برسد.

```
----------Step 1----------
Algorithm : ScheduleTheCalls(ListOfInstructions:List)
Begin
  Do
  Begin
1   Find a CallInstruction with the longest execution time from
    ListOfInstructions which is not selected.
2   Find all the parent nodes of the found CallInstruction in Data
    Dependency and Control Dependency Graph and put them in
    the NewListOfInstructions .
3   ScheduleTheCalls(NewListOfInstructions).
  End
  Until there is no CallInstruction
4 while there is a CommonInstruction in ListOfInstructions which
    is not selected
  Begin
5   Select a CommonInstruction ,Select the control dependent
    parents of the selected instruction before it if not selected
    before and then ResolveDependency for the selected
    instruction
  End
6 while there is a UseInstruction in ListOfInstructions which is
    not selected
  Begin
7   Select a UseInstruction ,Select the control dependent parents of
    the selected instruction before it if not selected before and
    then ResolveDependency for the selected instruction
  End
End
----------Step 2----------
Algorithm : SelectUseInstructions
Begin
1 while there is a UseInstruction which is not selected
  Begin
2   Select a UseInstruction ,Select the control dependent parents of
    the selected instruction before it if not selected before and
    then ResolveDependency for the selected instruction
  End
End
```

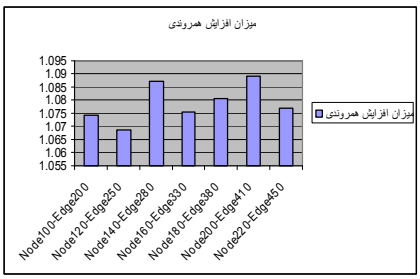
شکل (۲): مراحل ۱ و ۲ از الگوریتم جابجایی دستورات

```
----------Step 3----------
Algorithm :
ScheduleInstructionsFillWaitingTimes(ListOfUseInstructions)
Begin
1   while there is an Instruction in ListOfUseInstructions which is
        not selected
  Begin
2   Find the UseInstruction with the longest wait time from
    ListOfUseInstructions
3   Remove the UseInstruction form ListOfUseInstructions
4   Find all the candidate places between the UseInstruction and
    its corresponding CallInstruction and put them in
    CandidatePlacesList
5   For each place in CandidatePlacesList, if it is possible to put
    an instruction which is not selected
    Begin
6   Select the instruction ,Select the control dependent parents of
    the selected instruction before it if not selected before and
    then ResolveDependency for the selected instruction
7   Subtract the time of the instruction form the wait time of the
    UseInstruction.
8   If the wait time of the UseInstruction is equal to 0 then go to 1.
    End
  End
9 while there is an instruction which is not selected
  Begin
10  Select an instruction ,Select the control dependent parents of
    the selected instruction before it if not selected before and
    then ResolveDependency for the selected instruction
  End
End
----------Step 4----------
Algorithm: MoveInstructionBeforWait (ListOfUseInstructions)
Begin
1   while there is an Instruction in ListOfUseInstructions which is
        not selected
  Begin
2   Find the UseInstruction with the longest wait time from
    ListOfUseInstructions.
3   Remove the UseInstruction form ListOfUseInstructions
4   Add All the instruction of the scope which are placed after the
    UseInstruction to MovingList.
5   For each instruction in MovingList, if it is possible to move it
    before the UseInstruction
    Begin
6       Move the instruction to the place before the
        UseInstruction.
7       Subtract the time of the instruction form the wait time of
        the UseInstruction.
8       If the wait time of the UseInstruction is equal to 0 then
        go to 1.
    End
  End
End
```

شکل (۳): مراحل ۳ و ۴ از الگوریتم جابجایی دستورات




**شکل ۴. مقایسه زمان اجرای همروند به ازای ورودی های مختلف**




**شکل ۵. میزان افزایش همروندی پس از اعمال الگوریتم برای ورودیهای مختلف**

## مراجع

[1] John Hennessy and David Patterson, *Computer Architecture: A Quantitative Approach (Third Edition),* Morgan Kaufmann Publishers, 2003.

[2] A. Alet´a, J. Codina, J. S´anchez, A. Gonz´alez, and D. Kaeli. Exploiting pseudo-schedules to guide data dependence graph partitioning. In Procedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques, Charlottesville, VA, September 2002.

[3] B. Rau. Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops. In Proc. of 27th Int. Symp. on Microarchitecture, pages 67–74, Nov 1994.

[4] A. Eichenberger, E. Davidson, and S. Abraham. Optimum Module Schedules for Minimum Register Requirements. In Proc. of Supercomputing ’95, 1995.

[5] J. Llosa, M. Valero, E. Ayguad´e, and A. Gonz´alez. Modulo Scheduling with Reduced Register Pressure. IEEE Transactions on Computers, 47(6):625–638, 1998.

[6] Saeed Parsa, Vahid Khalilpour, ˝Automatic Distribution of serialized programs Using Java Symphony middleware”, LECTURE NOTES IN COMPUTER SCIENCE, (January 2006), SPRINGER-VERLAG.

[7] M. Corti and T. Gross. Instruction duration estimation by partial trace evaluation. In Proc. WIP Session 10th Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004. IEEE

[1] Instruction Scheduling