Archive of SID

روشی کارآمد برای تعریف توابع چندمتغیره با الگوهای عبارت برای آرایه‌ها در زبان سی‌پلاس‌پلاس و کودا

حسین محمودی داریان[1,*]

اطلاعات مقاله

دریافت مقاله: ۱۳۹۶/۰۱/۲۲
پذیرش مقاله: ۱۳۹۶/۱۱/۱۷

واژگان کلیدی:
الگوهای عبارت،
الگوهای متغیر،
سی‌پلاس‌پلاس،
پردازنده‌های گرافیکی،
کودا.

چکیده

در این مقاله یک روش کارآمد برای تعریف توابع چندمتغیره با الگوهای عبارت برای محاسبات آرایه‌ای در زبان برنامه‌نویسی سی‌پلاس‌پلاس جهت استفاده در شبیه‌سازی‌های دینامیک سیالات محاسباتی معرفی می‌شود. پیاده‌سازی روش با استفاده از الگوهای متغیر صورت می‌گیرد که از قابلیت‌های جدید زبان سی‌پلاس‌پلاس است. سادگی استفاده برای کاربران در زمینه‌های محاسباتی، از مزایای روش است، به‌طوری‌که کاربر بدون نیاز به اطلاع از مفاهیم برنامه‌نویسی، با الگوها به‌راحتی می‌تواند تابع خود را با هر تعداد آرگومان ورودی تعریف و سپس استفاده کند. روش حاضر می‌تواند جایگزین الگوهای عبارت مرسوم در توسعه کتابخانه‌های عددی باشد. برای سه تابع مختلف، شامل عملیات حسابی و توابع مثلثاتی، کارایی روش ارائه‌شده با روش الگوهای عبارت مرسوم، دو نحو مختلف زبان سی‌پلاس‌پلاس و زبان فرترن برای آرایه‌هایی با اندازه‌های مختلف، مقایسه می‌شود. به‌علاوه کارایی روش از لحاظ زمان ترجمه و حجم فایل اجرایی با مترجم‌های مختلف نشان داده می‌شود. مقایسه مشابهی روی پردازنده‌های گرافیکی با زبان کودا صورت می‌گیرد و کارایی روش، نشان داده می‌شود. نتایج، بیانگر آن است که روش حاضر برای هر تعداد درایه به لحاظ زمان محاسبات، زمان ترجمه و حجم فایل اجرایی کارایی بسیار خوبی دارد. در نهایت، به‌عنوان کاربردی از روش پیشنهادی، یک شبیه‌سازی عددی انجام می‌گیرد.

۱- مقدّمه

در شبیه‌سازی‌های عددی، سرعت انجام محاسبات، بسیار مهم است. تا دو دهه اخیر، زبان برنامه نویسی فرترن، انتخاب اول محققان برای توسعه کُدهای شبیه‌سازی با حجم محاسبات بالا بود، اما با پیشرفت‌های حاصل در زبان برنامه‌نویسی سی‌پلاس‌پلاس و قابلیت‌هایی که به‌تدریج به آن اضافه شد، این زبان نیز به یک انتخاب ایدئال برای کُدهای محاسباتی تبدیل گردید. برای مثال می‌توان به نرم‌افزارهای شبیه‌سازی متن-باز قدرتمندی نظیر اُپن‌فوم [۱،۲] و اُورچِر [۳،۴] (بر مبنای روش حجم محدود و تفاضل محدود) ودیل ۲ [۵،۶]، دیون [۷-۹]، فِنیکس [۱۰،۱۱]، نکتارپلاس‌پلاس [۱۲] (بر مبنای روش اجزای محدود) اشاره کرد که همگی با سی‌پلاس‌پلاس توسعه یافته‌اند. مهم‌ترین و شناخته‌شده‌ترین قابلیت زبان برنامه‌نویسی سی‌پلاس‌پلاس، برنامه‌نویسی شیءگرا است که این زبان را برای توسعه انواع گوناگونی از نرم‌افزارها قادر می‌سازد. در سی‌پلاس‌پلاس، قابلیت شیءگرایی با تعریف کلاس‌ها صورت می‌پذیرد. با این حال، استفاده از تمام امکانات برنامه‌نویسی شیءگرا برای توسعه یک کُد شبیه‌سازی عددی مناسب نیست. به‌عبارت‌دیگر، با استفاده از تمام امکانات برنامه‌نویسی شیءگرا می‌توان یک کُد شبیه‌سازی قدرتمند تولید کرد که خصوصیات مثبتی نظیر سادگی توسعه کُد جهت افزودن مدل‌های جدید، سادگی استفاده برای کاربران و حتی روان بودن متن کُد، داشته

* پست الکترونیک نویسنده مسئول: hmahmoodi@ut.ac.ir
۱. استادیار، دانشکده علوم مهندسی، دانشکده فنی، دانشگاه تهران

www.SID.ir

Archive of SID

باشد؛ اما این خصوصیات مثبت از سوی دیگر می‌تواند موجب کاهش کارایی محاسباتی کُد (سرعت اجرای شبیه‌سازی) شود.

یکی دیگر از قابلیت‌های مهم زبان برنامه‌نویسی سی‌پلاس‌پلاس، الگوها هستند. قدرت الگوها شاید در ابتدا برای توسعه‌دهندگان آن نیز مشخص نبود. با شناخته‌شدن تدریجی توانایی الگوها حتی مفاهیم جدید نظیر فرابرنامه‌نویسی الگوها پدید آمد. در این زمینه، خواننده علاقه‌مند به مرجع [۱۳] ارجاع داده می‌شود. استفاده هم‌زمان از الگوها و کلاس‌ها مزایای فراوانی ایجاد می‌کند که بسیاری از آن‌ها در کتابخانه الگوی استاندارد گنجانده شده است. امکان فوق‌العاده‌ای که الگوها ایجاد می‌کنند، امکان الگوهای عبارت است. این امکان در کتابخانه‌های محاسباتی جبر خطی، نظیر پوما، بوست یوبلَس و بلتیز پلاس‌پلاس و همچنین در نرم‌افزارهای متن-بازی که پیش‌تر ذکر شد، استفاده شده است [۱۴-۱۶].

به‌طور خلاصه، الگوهای عبارت، این امکان را فراهم می‌کند که نحو برنامه‌نویسی عملیات ریاضی (نظیر چهار عملی اصلی) برای آرایه‌ها مشابه نوشتار ریاضی دست‌نویس باشد، بدون آنکه سرعت انجام محاسبات کاهش یابد. شایان ذکر است نکته اصلی، کاهش نیافتن سرعت محاسبات است؛ زیرا بسیار پیش از این نیز با استفاده از کلاس‌ها امکان حفظ مشابهت نحو برنامه‌نویسی با نوشتار ریاضی وجود داشت. امروزه چهار عملی اصلی و توابع استاندارد ریاضی، نظیر توابع مثلثاتی با استفاده از الگوهای عبارت به کاربر این امکان را می‌دهد که عملیات برداری را به‌راحتی در کُد پیاده‌سازی کند، بدون اینکه شاهد افت کارایی محاسباتی باشد.

در کتابخانه‌های محاسباتی، تنها توابع استاندارد برای استفاده در الگوهای عبارت تعریف شده‌اند. اگر کاربر نیاز به تعریف تابعی دلخواه داشته باشد، باید از طراحی کتابخانه و مفاهیم مربوط به الگوهای عبارت به اندازه کافی آگاه باشد. به‌علاوه توابع استاندارد دارای تعداد اندکی آرگومان ورودی هستند. برای مثال تابع sin(x) یک آرگومان ورودی و تابع atan2(y,x) دارای دو آرگومان ورودی است. برای مثال، در کتابخانه بلیتزپلاس‌پلاس با استفاده از ماکروها راهی برای تعریف توابع یک و دو متغیره پیش‌بینی شده است؛ اما برای بیش از دو متغیر، تمهیدی وجود ندارد [۱۶].

در تحقیق حاضر، یک راه ساده و کارآمد برای تعریف توابع معرفی می‌شود تا کاربر بتواند تابع دلخواه خود را با هر تعداد آرگومان ورودی تعریف کند، بدون اینکه نیاز به دانستن مفاهیم الگوهای عبارت داشته باشد. برای این کار از قابلیت الگوهای متغیر، از قابلیت‌های جدید سی‌پلاس‌پلاس، استفاده شده است. همچنین ازآنجاکه روش الگوهای عبارت، یک راه مناسب برای استفاده از قابلیت‌های پردازنده‌های گرافیکی است، در اینجا نشان داده می‌شود که روش ارائه‌شده به‌سادگی با این پردازنده‌ها نیز قابل استفاده است.

بخش ۲ شامل چندین زیربخش است که به شرح ایده الگوهای عبارت می‌پردازد. در زیربخش ۲-۱ سربارگذاری عملگرها به‌صورت مرسوم به اختصار معرفی می‌شود و در زیربخش ۲-۲ نحوه سربارگذاری عملگرهای حسابی (چهار عمل اصلی) در الگوهای عبارت شرح داده می‌شود. زیربخش ۲-۳ توسعه الگوهای عبارت را برای توابع ریاضی شرح می‌دهد. بخش ۳ اختصاص به معرفی ایده اصلی کار حاضر، توابع چندمتغیره در چارچوب الگوهای عبارت، دارد. در بخش ۴ نتایج و کارایی روش ارائه، از لحاظ زمان محاسباتی و زمان ترجمه با مترجم‌های مختلف ارائه و با سایر روش‌ها مقایسه می‌شود و همچنین یک شبیه‌سازی عددی با روش پیشنهادی انجام می‌گیرد. در انتها نیز نتیجه‌گیری قرار دارد.

۲- الگوهای عبارت

۲- ۱- سربارگذاری عملگرها به‌صورت مرسوم

برای شروع، کلاسی به نام arr تعریف می‌کنیم که شامل آرایه‌ای با N درایه است (شایان ذکر است برای رعایت اختصار، از آرایه‌های ایستا استفاده می‌کنیم؛ اما در بخش نتایج، آرایه‌ها پویا خواهند بود):

نوشته ۱

```
const int N = 100;
class arr
{
public:
        double a[N];
};
```

با استفاده از سربارگذاری عملگرها و توابع می‌توان محاسبات برداری را مشابه نحو ریاضی آن‌ها در برنامه پیاده‌سازی کرد [۱۷]. در نوشته ۲ محاسبات آرایه‌ای برای یک عبارت ریاضی به‌صورت عادی (قسمت ۱) و سپس با

www.SID.ir

Archive of SID

استفاده از سربارگذاری عملگرها (قسمت ۲) نشان داده شده است:

نوشته ۲

```
int main()
{
	arr A, B, C;

	// 1- Without overloading
	for ( int i = 0; i < N ; i++ )
	{
	C.a[i] = B.a[i] * A.a[i]
		+ B.a[i] –  A.a[i];
	}

	// 2- Using operator overloading
	C = B * A + B - A;
	return 0;
};
```

با استفاده از سربارگذاری عملگرها، برنامهٔ نوشته‌شده، قابل فهم‌تر و خلاصه‌تر است، اما این سربارگذاری یک مشکل کاملاً شناخته‌شده دارد که جز برای مقادیر خیلی کوچک N استفاده نمی‌شود. مشکل اینجا است که در قسمت ۲ به‌ازای هر عملگر، کامپیوتر حین اجرا یک متغیر موقت از نوع arr برای ذخیره‌سازی حاصل آن عملگر در حافظه ایجاد می‌کند و پس از اتمام اجرای عملگر، آن متغیر را از حافظه پاک می‌کند. هرچه N بزرگ‌تر باشد، این تخصیص و آزادسازی حافظه، زمان بیشتری طول می‌کشد، درحالی‌که در قسمت ۱ چنین کاری صورت نمی‌گیرد. از سوی دیگر، هرچه در یک عبارت، تعداد عملگرها بیشتر باشد، حافظه موقت بیشتری نیاز است. به همین علت قسمت ۱ هم به‌لحاظ سرعت محاسبات و هم به‌لحاظ اشغال کمتر حافظه، ارجحیت دارد.

۲- ۲- سربارگذاری عملگرها با الگوهای عبارت

با استفاده از الگوهای عبارت می‌توان مشکل مذکور را حل کرد. در این روش، کلاسی برای عبارت‌های دوتایی تعریف می‌شود، سپس هر عملگر، نظیر جمع، به گونه‌ای سربارگذاری می‌شود که یک متغیر از نوع عبارت دوتایی شامل عملوندهای چپ و راست باز می‌گرداند. نوشته زیر، تعریف کلاس دوتایی با نام BinExpr و نحوه سربارگذاری عملگر جمع را نشان می‌دهد.

مشاهده می‌شود کلاس BinExpr یک کلاس الگویی با سه نوع L، R و Op است.

نوشته ۳

```
template<typename L, typename R, typename
Op>
class BinExpr
{
	L &l;
	R &r;
public:
	double eval(int i)
	{
		return Op::apply(l.eval(i),
r.eval(i));
	}
	BinExpr(L &_l_, R &_r_) :l(_l_), r(_r_){}
};

class Add
{
public:
	static double apply(real l, real r)
	{
		return l + r;
	}
};

template<typename L, typename R>
BinExpr<L, R, Add> operator+(L &l, R &r)
{
	return BinExpr<L, R, Add>(l, r);
}
```

L و R به‌ترتیب، نوع عملوندهای چپ و راست و Op نوع عملگر عبارت است. به همین علت، نوع خروجی سربارگذاری عملگر جمع به‌صورت BinExpr<L, R, Add> است. کلاس Add تنها شامل یک تابع ایستا است که حاصل جمع دو عدد را باز می‌گرداند. تا اینجا نوشتن A+B محاسباتی انجام نمی‌دهد. در حقیقت، حاصل این جمع، متغیری از نوع BinExpr<arr, arr, Add> است. برای اینکه جمع درایه‌های متناظر انجام شود، اصلاحاتی در کلاس arr نیاز است که مهم‌ترین آن، سربارگذاری عملگر انتساب (=) خواهد بود:

تابع eval نیز مقدار درایه iاُم آرایه را باز می‌گرداند. حال هنگامی که کاربر داخل برنامه عبارت C = A+B را می‌نویسد، مترجم آن را به‌صورت خودکار طی مراحل زیر باز می‌کند:

www.SID.ir

Archive of SID

نوشته ۴

```
const int N = 100;
class arr
{
public:
        double a[N];
        double eval(inti)
        {
                return a[i];
        }

        template<typename R>
        void operator=(R&expr)
        {
                for ( int i = 0; i < N ; i++ )
                {
                        a[i] = expr.eval(i);
                }
        }
};
```

```
1. C = A + B;
2. C = BinExpr<arr,arr,Add>(A,B)
```

سپس عملگر انتساب باعث می‌شود متغیر C تابع زیر را فراخوانی کند:

```
void operator=(BinExpr<arr,arr,Add>&expr)
{
        for ( int i = 0; i < N ; i++ )
        {
                a[i] = expr.eval(i);
        }
}
```

که دوباره مترجم، عبارت داخل حلقه را به‌صورت خودکار طی مراحل زیر باز می‌کند:

```
1. a[i] = r.eval(i);
2. a[i] = Add::apply(expr.l.eval(i),
expr.r.eval(i));
3. a[i] = Add::apply(expr.l.a[i], expr.r.a[i]);
4. a[i] = expr.l.a[i] + expr.r.a[i];
```

به‌عنوان یک مثال دیگر، باز شدن عبارت زیر را که دو عمل جمع را دربردارد، در نظر می‌گیریم:

```
1. C = A + B + A;
2. C = BinExpr<arr,arr,Add>(A,B) + A;
3. C = tmp + A;
4. C = BinExpr<
BinExpr<arr,arr,Add>,arr,Add>(tmp,A);
```

و سپس عبارت داخل حلقه در عملگر انتساب به‌صورت زیر باز می‌شود:

```
1. a[i] = r.eval(i);
2. a[i] = Add::apply(expr.l.eval(i), expr.r.eval(i));
3. a[i] = Add::apply(Add::apply(expr.l.l.eval(i),
expr.l.r.eval(i)) , expr.r.a[i]);
4. a[i] = Add::apply(Add::apply(expr.l.l.a[i],
expr.l.r.a[i]) , expr.r.a[i]);
5. a[i] = Add::apply(expr.l.l.a[i] + expr.l.r.a[i]) ,
expr.r.a[i]);
6. a[i] = ( expr.l.l.a[i] + expr.l.r.a[i] ) + expr.r.a[i];
```

درحقیقت، تنها عبارت $C = A + B + A$ در برنامه نوشته می‌شود و باقی آن را مترجم حین ساختن فایل اجرایی (پنهان از دید کاربر) انجام می‌دهد. مشاهده می‌شود که با استفاده از الگوهای عبارت، هیچ حافظه موقتی برای انجام عملیات تخصیص نمی‌یابد و عملیات ریاضی همانند نحو عادی (نوشته ۲– قسمت ۲) انجام می‌گیرد.

۲- ۳- الگوهای عبارت برای توابع ریاضی

برای توابع ریاضی دومتغیره می‌توان از همان کلاس BinExpr استفاده کرد. برای مثال، تابع استاندارد توان pow(x,y) را در نظر می‌گیریم که در آن، عدد x به توان عدد y می‌رسد. برای اینکه بتوان این تابع را برای کلاس arr نیز به کار برد، ابتدا آن را به‌صورت ایستا درون یک کلاس می‌گنجانیم:

```
class Power
{
public:
        static double apply(real l, real r)
        {
                return pow(l,r);
        }
};
```

سپس تابع pow را به‌صورت زیر سربارگذاری می‌کنیم:

www.SID.ir

Archive of SID

```
template<typename L, typename R>
BinExpr<L, R, Power> pow(L &l, R &r)
{
	return BinExpr<L, R, Power>(l, r);
}
```

در این صورت، کاربر کافی است عبارت زیر را بنویسد:

```
C = pow(A, B);
```

و سپس بقیهٔ عملیات، شامل به توان رساندن تک‌تک درایه‌های A به درایه متناظر B و ذخیرهٔ آن در درایه متناظر C، برعهدهٔ مترجم است. برای سایر توابع استاندارد ریاضی نیز به نحو مشابه عمل می‌شود. به طور خاص برای توابع یک‌متغیره نظیر sin و log ابتدا باید کلاسی مشابه BinExpr تعریف کرد که تنها دربرگیرندهٔ یک عبارت باشد، همانند کلاس UnaExpr در برنامهٔ زیر که برای مثال، تابع sin در آن سربارگذاری شده است:

نوشته ۵

```
template<typename L, typename Op>
class UnaExpr
{
	L &l;
public:
	double eval(int i)
	{
		return Op::apply(l.eval(i));
	}
	UnaExpr(L &_l_) :l(_l_){}
};

class sinExpr
{
public:
	static double apply(real l)
	{
		return sin(l);
	}
};

template<typename L>
UnaExpr<L, sinExpr> sin(L &l)
{
	return UnaExpr<L, sinExpr>(l, r);
}
```

شایان ذکر است، استفاده از الگوها افزایش زمان ترجمه را دربردارد؛ با این حال، در شبیه‌سازی‌های عددی، زمان اجرا بسیار مهم‌تر از زمان ترجمه است. مزیت دیگر استفاده از الگوهای عبارت نسبت به سربارگذاری مرسوم این است که در سربارگذاری مرسوم، عملگرها یا توابع درون یک حلقه برای تمام مؤلفه‌های آرایه محاسبه می‌شوند. درحقیقت، حلقه و رابطهٔ ریاضی یکپارچه هستند [۱۷]. اما با استفاده از الگوها، رابطهٔ ریاضی و حلقه از یکدیگر جدا هستند و کاربر تنها رابطهٔ ریاضی را تعریف می‌کند و تولید حلقه برعهدهٔ مترجم است. همچنین جدا بودن رابطهٔ ریاضی و حلقه، این امکان را فراهم می‌کند که بتوان رابطهٔ ریاضی را درون حلقه‌های مختلف قرار داد و محاسبهٔ رابطهٔ ریاضی، تنها برای بعضی از درایه‌های آرایه انجام شود یا اینکه حلقه با امکانات مختلف نظیر اُپن‌اِم‌پی یا کودا اجرا گردد.

۳- الگوهای عبارت برای توابع چندمتغیره

در قسمت‌های قبل دیدیم که چگونه چهار عمل اصلی و توابع استاندارد ریاضی برای استفاده برداری با به‌کاربردن الگوهای عبارت در کتابخانه‌ها گنجانده می‌شود و کاربر با اضافه کردن آن کتابخانه به برنامه می‌تواند از تمام امکانات آن استفاده کند. اما اگر کاربر نیاز داشته باشد که یک تابع دلخواه تعریف و از آن استفاده کند، باید اطلاعات بیشتری از کتابخانه عددی و دانش برنامه‌نویسی داشته باشد. از سوی دیگر، با استفاده از کلاس UnaExpr و BinExpr تنها توابع یک و دو متغیره می‌توان تعریف کرد. در اینجا راهی معرفی می‌کنیم تا کاربر با استفاده از آن به‌سادگی، تابع دلخواه خود را با تعداد دلخواه متغیر تعریف و از آن برای عملیات برداری استفاده کند.

پیش از شرح طراحی، نشان می‌دهیم که کاربر چگونه به‌سادگی، تابع خود را تعریف و از آن استفاده می‌کند. فرض کنید کاربر نیاز به تعریف تابعی برای محاسبه طول در فضای سه‌بُعدی دارد. برای این کار، ابتدا تابع به‌صورت زیر تعریف می‌شود:

```
struct Length : public MultiVarFun<Length>
{
	static real apply(real x, real y, real z)
	{
		return sqrt(x*x+y*y+z*z);
	}
};
```

درحقیقت، کاربر تابع خود را درون یک کلاس قرار می‌دهد.

www.SID.ir

Archive of SID

تابع درون کلاس باید ایستا و نام آن نیز apply باشد. این کلاس از کلاس دیگری به نام MultiVarFun<Length> ارث می‌برد. این کلاس که جزئی از طراحی است، در ادامه شرح داده خواهد شد. داخل برنامه تابع به‌صورت زیر برای بردارها فراخوانی می‌شود:

```
arr A,B,C,D;
...
D = Length::func(A, B, C);
```

مشاهده می‌شود که تقریباً کاربر مشابه تعریف یک تابع معمولی در سی‌پلاس‌پلاس می‌تواند تابع دلخواه خود را تعریف و استفاده کند.

طراحی با استفاده از تعریف دو کلاس به نام‌های MultiExpr و MultiVarFun است. کلاس MultiExpr کلاس اصلی و مشابه دو کلاس UnaExpr و BinExpr است، با این تفاوت که چندین عبارت را شامل می‌شود. قابلیت الگوهای متغیر، این امکان را فراهم می‌سازد که کلاسی با تعداد نامعلومی پارامتر تعریف کنیم:

نوشته ۶

```
template<typename Op, typename ExprType,
typename... Args >
class MultiExpr :MultiExpr<Op, Args... >
{
	ExprType& L;
public:
	real eval(int i)
	{
		return MultiExpr<Op, Args...
>::f(i, L);
	}
	template<typename... Args2 >
	real f(int i, Args2&... args2)
	{
		return MultiExpr<Op, Args...
>::f(i, args2..., L);
	}
	MultiExpr(ExprType& _l_, Args&...
_args_) :L(_l_), MultiExpr<Op, Args...
>(_args_...){}
};

template<typename Op, typename ExprType>
class MultiExpr<Op, ExprType>
{
	ExprType& L;
public:
	real eval(int i)
	{
		return Op::apply(L.eval(i));
	}
	template<typename... Args >
	real f(int i, Args&... args)
	{
		return
Op::apply(args.eval(i)..., L.eval(i) );
	}
	MultiExpr(ExprType& _l_) :L(_l_){}
};
```

کلاس MultiExpr به‌صورت ارث‌بری بازگشتی تعریف شده است. درحقیقت، پدر هر کلاسی، کلاسی با یک پارامتر کمتر است. در نهایت، کلاسی که در بالای درخت ارث‌بری قرار دارد، کلاس MultiExpr<Op, ExprType> است که پایان‌دهندهٔ ارث‌بری است. شکل (۱) درخت ارث‌بری را برای عبارت Length::fun(A, B, C) نشان می‌دهد. تابع سازنده کلاس هر سه آرایه را می‌گیرد، سپس آرایه سمت چپ را به متغیر عضو L خود نسبت می‌دهد و بقیهٔ آرایه‌ها را به تابع سازنده کلاس پدر واگذار می‌کند. بدین ترتیب، آرایه‌ای که در سمت چپ سایر آرایه‌ها قرار دارد، عضو پایین‌ترین فرزند و آرایه‌ای که در سمت راست سایر آرایه‌ها قرار دارد، عضو بالاترین پدر خواهد بود. شکل (۱) این مطلب را برای مثال اخیر نشان می‌دهد.

هنگام محاسبه عبارت که تابع eval() آن را انجام می‌دهد، ابتدا این تابع، تابع f() متعلق به پدر را با اندیس i و آرایه عضو خود (L) فراخوانی می‌کند. سپس به نحو مشابه، تابع f() آرایه عضو کلاس خود را سمت چپ آرایه‌های دریافتی از کلاس فرزند همراه با اندیس i به کلاس پدر خود واگذار می‌کند. به عنوان مثال، در شکل (۲) درخت فراخوانی تابع eval() برای تابع طول مشاهده می‌شود.

شایان ذکر است، با وجود کلاس MultiExpr دیگر نیازی به دو کلاس BinExpr و UnaExpr نیست و با استفاده از آن می‌توان عملگرها و تمام توابع استاندارد را تعریف کرد.

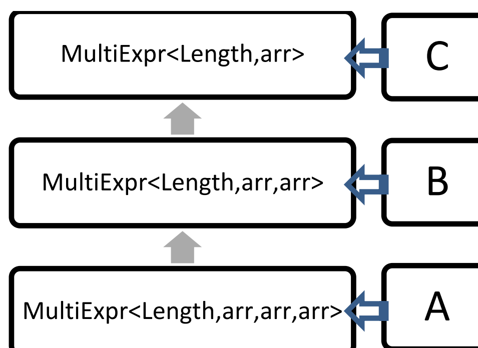
۴- نتایج

در اینجا نتایج زمان‌سنجی و کارایی ایدهٔ معرفی‌شده را بررسی می‌کنیم. سه تابع مختلف در نظر می‌گیریم و زمان اجرای هریک را با استفاده از روش‌های مختلف ارائه می‌نماییم. همچنین زمان اجرا با استفاده از نرم‌افزار فرترن

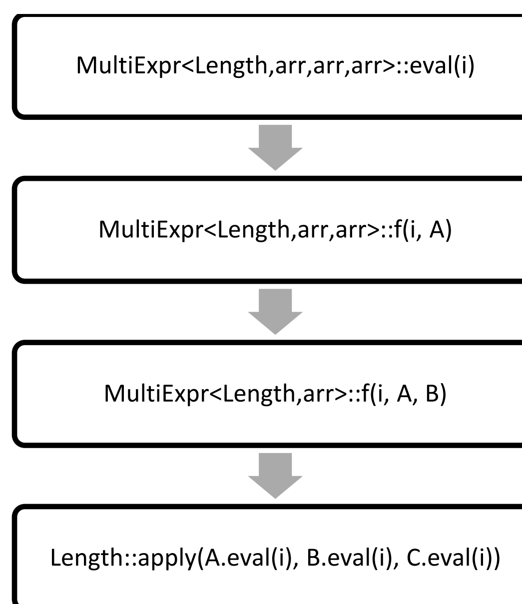
www.SID.ir

Archive of SID

ارائه می‌گردد که در شبیه‌سازی‌های سیالاتی بسیار استفاده می‌شود [۱۸-۲۰]. پردازنده مرکزی استفاده‌شده Intel Core 2 Quad Q6600 و پردازنده گرافیکی NVidiaGeForce 560 GTX است.



شکل ۱: نمودار ارث‌بری و نحوه عضویت آرایه‌ها در کلاس‌ها برای عبارت Length::fun(A, B, C)



شکل ۲: درخت فراخوانی تابع eval(i) برای عبارت Length::fun(A, B, C)

برای زمان‌سنجی توابع زیر استفاده می‌شوند:

$$g(x.y.z.v) = xy + zv$$

$$f(x) = x + xx + xxx + xxxx$$

$$h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$$

تابع دوم، تابعی یک‌متغیره و دو تابع دیگر چهارمتغیره هستند. علت انتخاب تابع دوم این است که یکی از ضعف‌های الگوهای عبارت هنگامی است که یک متغیر چندین بار در یک عبارت ریاضی ظاهر می‌شود [۲۱]. در اینجا بررسی می‌کنیم که آیا این ضعف با استفاده از ایده حاضر باقی می‌ماند یا خیر.

۴-۱- زمان‌سنجی با پردازنده اصلی (پردازنده مرکزی)

زمان‌سنجی برای آرایه‌هایی با تعداد درایه‌های $N=10$ تا $N=10^7$ ارائه می‌شود. البته در تعریف کلاس arr آرایه پویا جایگزین آرایه ایستا می‌شود؛ زیرا در شبیه‌سازی‌های عددی شبکه عددی اغلب با آرایه پویا تخصیص حافظه می‌شود. در ابتدا آرایه‌های زیر تعریف می‌شوند:

```
arr A, B, C, D, R;
```

روش‌هایی که با آن‌ها زمان‌سنجی انجام می‌شود (برای نمونه برای تابع (g(x,y,z,v)$g(x.y.z.v)$، به شرح زیر است:

الف. با استفاده از نحو زبان سی و استفاده از اشاره‌گرها (یا آرایه‌ها): در این روش، اشاره‌گر موجود در هر شیء از نوع arr به یک اشاره‌گر درون برنامه منتسب می‌شود تا همانند آرایه‌های معمولی سی بتوان به درایه‌های آن دسترسی داشت.

```
double *a, *b, *c, *d, *r;
a = A.a; b = B.a; c = C.a; d = D.a; e = R.a;
for (i = 0; i < N; i++)
{
        r[i] = a[i]*b[i] + c[i]*d[i];
}
```

ب. مشابه روش ۱، با این تفاوت که اشاره‌گرها با عملگر افزایش (++) پس از محاسبه تابع به درایه بعدی منتقل می‌شود. البته این روش شاید برای برنامه‌نویسان حرفه‌ای در کاربردهای خاص که از دید کاربر پنهان است، بسیار مطلوب باشد، اما برای یک کُد محاسباتی، چندان مناسب نیست؛ زیرا هم خوانایی کُد را کاهش می‌دهد و هم احتمال بروز خطا در آن زیاد است (وقتی قرار باشد هر محاسباتی به این صورت انجام شود). با این حال، چون انتظار می‌رود با این روش زمان محاسبات کمتر باشد، از آن استفاده می‌کنیم.

www.SID.ir

Archive of SID

```
a = A.a; b = B.a; c = C.a; d = D.a; e = R.a;
for (i = 0; i < N; i++)
{
        *r = (*a)*(*b) + (*c)*(*d);
        r++; a++; b++; c++; d++;
}
```

ج. با استفاده از الگوهای عبارت مرسوم.

```
R = A*B + C*D;
```

د. با استفاده از الگوهای عبارت برای توابع چندمتغیره که درحقیقت، ایدهٔ معرفی‌شده در مقالهٔ حاضر است.

```
R = g::func(A, B, C, D);
```

ه. با زبان فرترن.

```
REAL(8), ALLOCATABLE :: A(:), B(:), C(:), D(:), R(:)
ALLOCATE(A(N),B(N),C(N),D(N),R(N))

DO I = 1,N
        R(I) = A(I)*B(I) + C(I)*D(I)
ENDDO
```

مقداردهی اولیه برای آرایه‌ها در تمام روش‌ها به‌صورت زیر است:

$$A_i = e^{13.6(i-N/2)/N}.\ B_i = 0.1A_i.\ C_i = 0.01A_i.\ D_i = 10A_i$$

در این صورت، درایه‌های آرایه‌ها از مقیاس‌های مختلفی خواهند بود. برای مثال، درایه‌های آرایه A از حدود ۰٫۰۰۱ تا ۲۳۰ هستند.

در شکل‌های (۳) تا (۵)، نتایج زمان‌سنجی‌ها برای هر ۵ روش نمایش داده شده‌اند. مترجم استفاده‌شده، مترجم ویژوال استودیو با پیکربندی ریلیس (پیکربندی بهینه) است. در شکل‌ها روش‌های الف تا ه به‌ترتیب، با pointer، ++pointer، ET، ET function و Fortran مشخص و زمان‌های اجرا بر تعداد درایه‌ها (N) تقسیم شده‌اند. راجع به زمان‌سنجی‌ها ذکر این نکته ضروری است که تمام توابع با تمام روش‌ها چندین بار تکرار شده‌اند تا از درستی نتایج ارائه‌شده، اطمینان حاصل شود.

شکل (۳) نتایج را برای تابع $g(x.y.z.v)$ نمایش می‌دهد. مشاهده می‌شود با استفاده از اشاره‌گرها (روش الف) و فرترن، بهترین نتایج حاصل شده است. سپس، نتایج روش ب و د بهتر هستند و در آخر، روش الگوهای عبارت مرسوم است که بیشترین زمان اجرا را دارد. اما برای Nهای بزرگ، تمام روش‌ها یکسان عمل کرده‌اند.

شکل (۴) نتایج را برای تابع یک‌متغیره $f(x)$ نشان می‌دهد. مشاهده می‌شود برای تعداد درایه‌های کم، غیر از روش الگوهای عبارت، سایر روش‌ها مشابه هستند. یکی از ایرادات روش الگوهای عبارت همین است که وقتی یک متغیر داخل یک عبارت، چند بار ظاهر می‌شود، مترجم نمی‌تواند تشخیص بدهد و در نتیجه، نمی‌تواند بهینه‌سازی لازم را انجام دهد. در حالی که وقتی آرایه‌ها به‌صورت عادی (روش الف) در عبارت ظاهر می‌شوند، مترجم، آرایه‌های یکسان را تشخیص می‌دهد. برای تعداد درایه‌های حدود 10^2 تا 10^5، فرترن کمترین زمان را دارد و روش‌های ب تا د یکسان هستند. جالب است که برای تعداد درایه‌های بیش از 10^6 روش الف ناگهان با افت کارایی مواجه می‌شود و همانند روش الگوهای عبارت، زمان صرف می‌کند. شایان ذکر است این افت در کارایی در شکل (۱) نیز به‌گونه‌ای مشاهده می‌شود. به‌علاوه برای این تعداد درایه، روش‌های ب و د عملکردی همانند فرترن در زمان محاسبات دارند.

شکل (۳) نتایج زمان‌سنجی برای تابع $h(x.y.z.v)$ را نشان می‌دهد. در این تابع، علاوه بر عملیات حسابی، توابع مثلثاتی نیز وجود دارند. مشاهده می‌شود روش‌های ب و د بهترین عملکرد را دارند. تابع فرترن نیز غیر از N = 10 همانند این دو روش عمل می‌کند. روش الگوهای عبارت همانند توابع قبل، بیشترین زمان را صرف می‌کند؛ اما روش الف برخلاف توابع قبل حتی در تعداد درایه‌های کوچک نیز کارایی مناسبی ندارد.

در مجموع می‌توان گفت، نه‌تنها استفاده از روش د کاراتر از روش ج است و معایب آن را ندارد، بلکه به بهترین عملکرد نزدیک است و حتی گاهی بهترین عملکرد را نیز دارد.

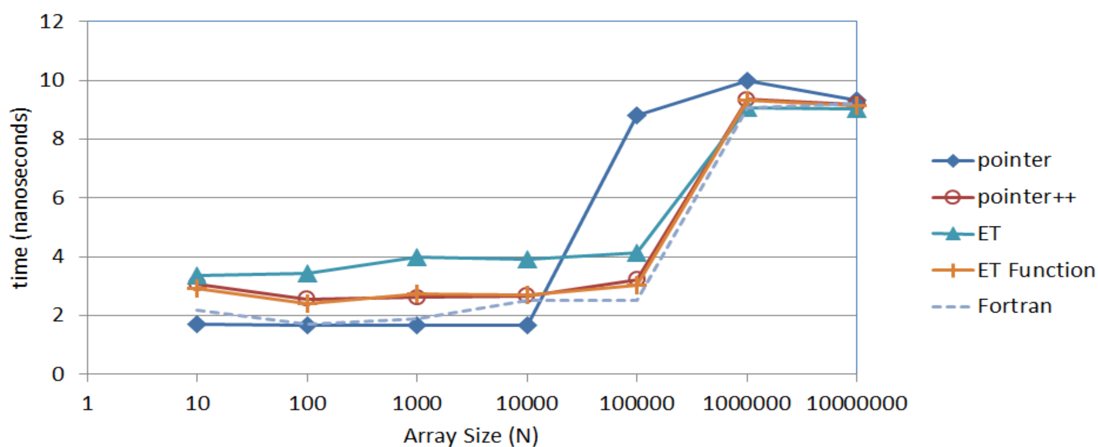
۴-۲- زمان‌سنجی با سایر مترجم‌ها

از آنجا که زمان اجرا وابسته به مترجم است، در این قسمت زمان‌سنجی با مترجم‌های جی‌سی‌سی، اِل‌اِل‌وی‌اِم و اینتل نیز ارائه می‌شود.

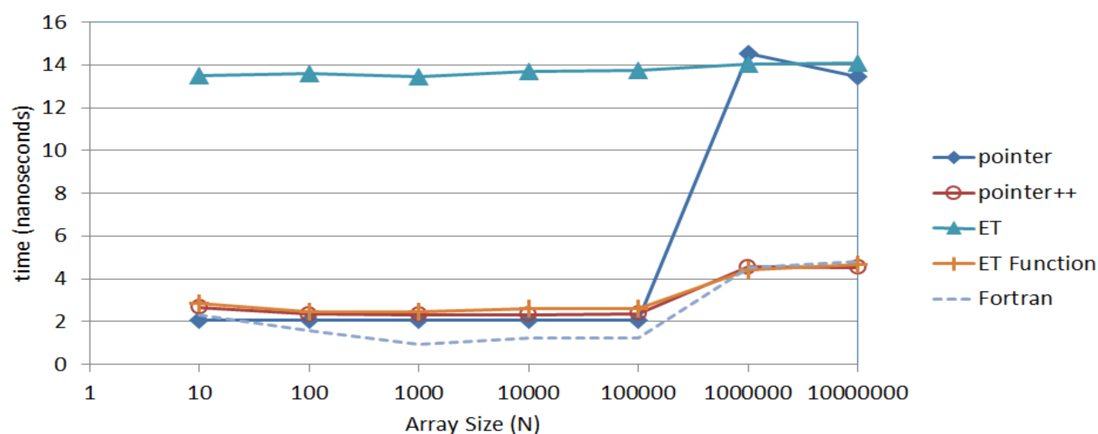
شکل‌های (۶) تا (۸) مقایسه زمان‌های اجرا را با استفاده از مترجم‌های مختلف نمایش می‌دهد. برای تمام مترجم‌ها گزینهٔ بهینه‌سازی (O2-) فعال است.

www.SID.ir

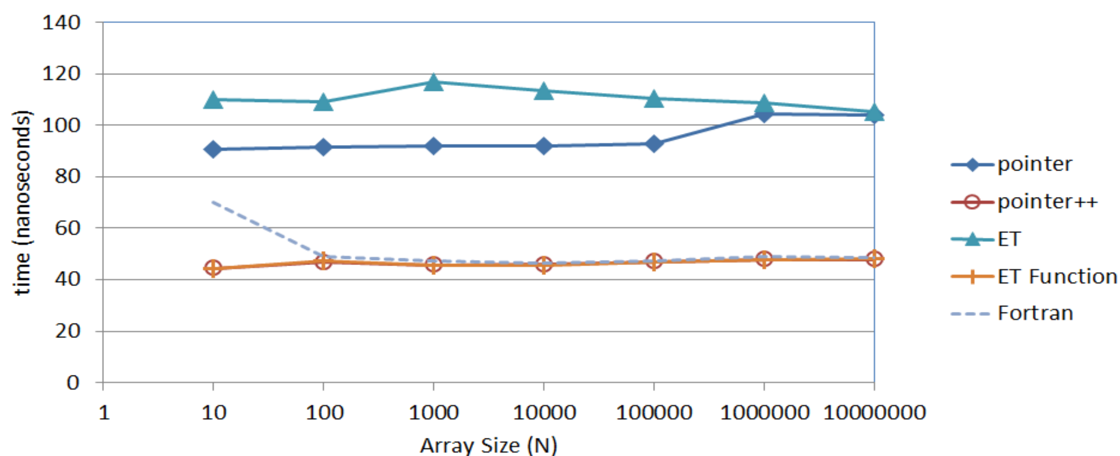
Archive of SID



شکل ۳: زمان‌سنجی تابع $g(x.y.z.v) = xy + zv$



شکل ۴: زمان‌سنجی تابع $f(x) = x + xx + xxx + xxxx$

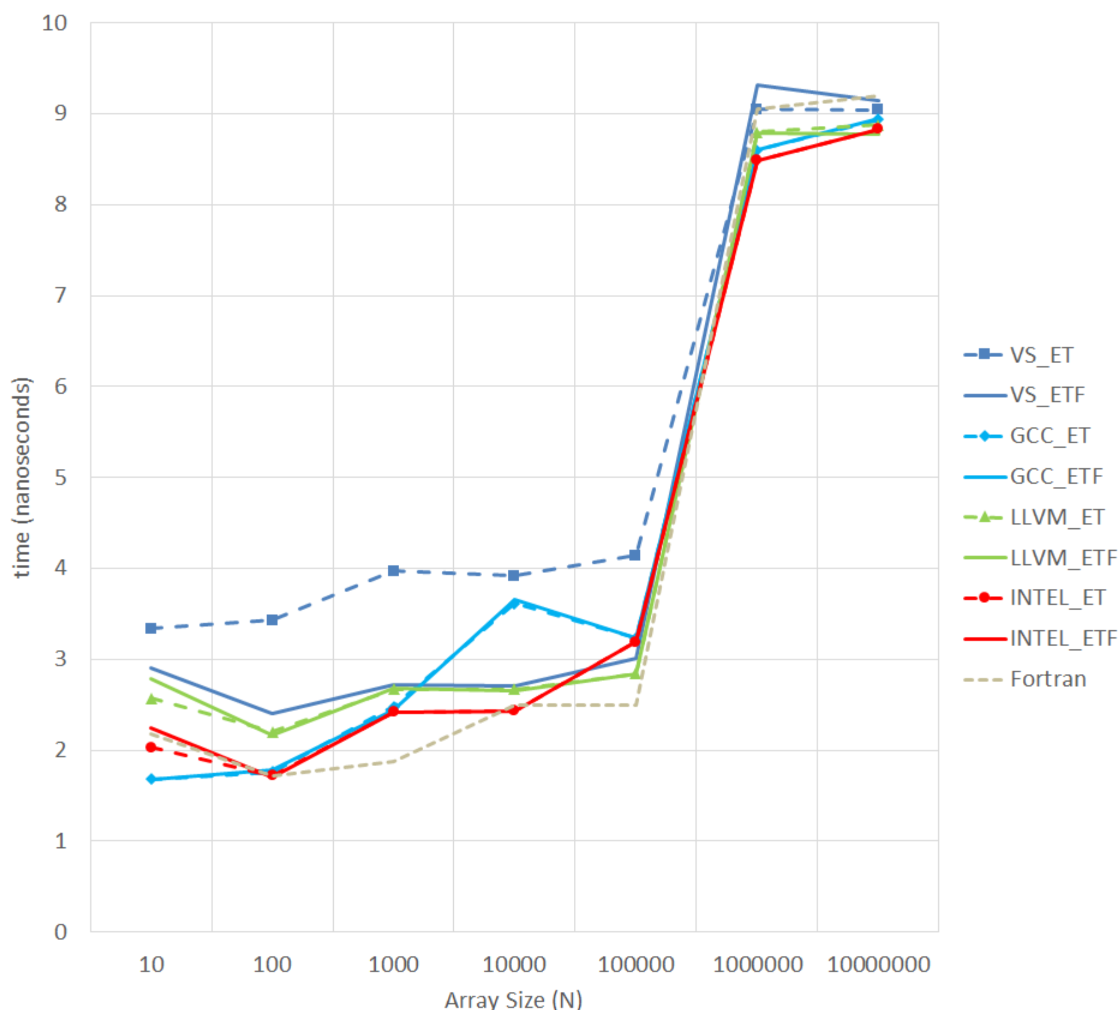


شکل ۵: زمان‌سنجی تابع $h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$

www.SID.ir

Archive of SID

در شکل‌های (۶) و (۷) برای توابع $g(x.y.z.v)$ و $f(x)$ مشاهده می‌شود که غیر از مترجم ویژوال استودیو، برای تمام مترجم‌ها تفاوتی میانی زمان اجرا با الگوهای عبارت مرسوم و زمان اجرا با الگوهای عبارت برای توابع چندمتغیره وجود ندارد. در شکل (۸) برای تابع $h(x.y.z.v)$ مشاهده می‌شود که برای تمام مترجم‌ها (غیر از جی‌سی‌سی) زمان اجرا با الگوهای عبارت برای توابع چندمتغیره (روش د) کمتر از زمان اجرا با الگوهای عبارت مرسوم (روش ج) است.



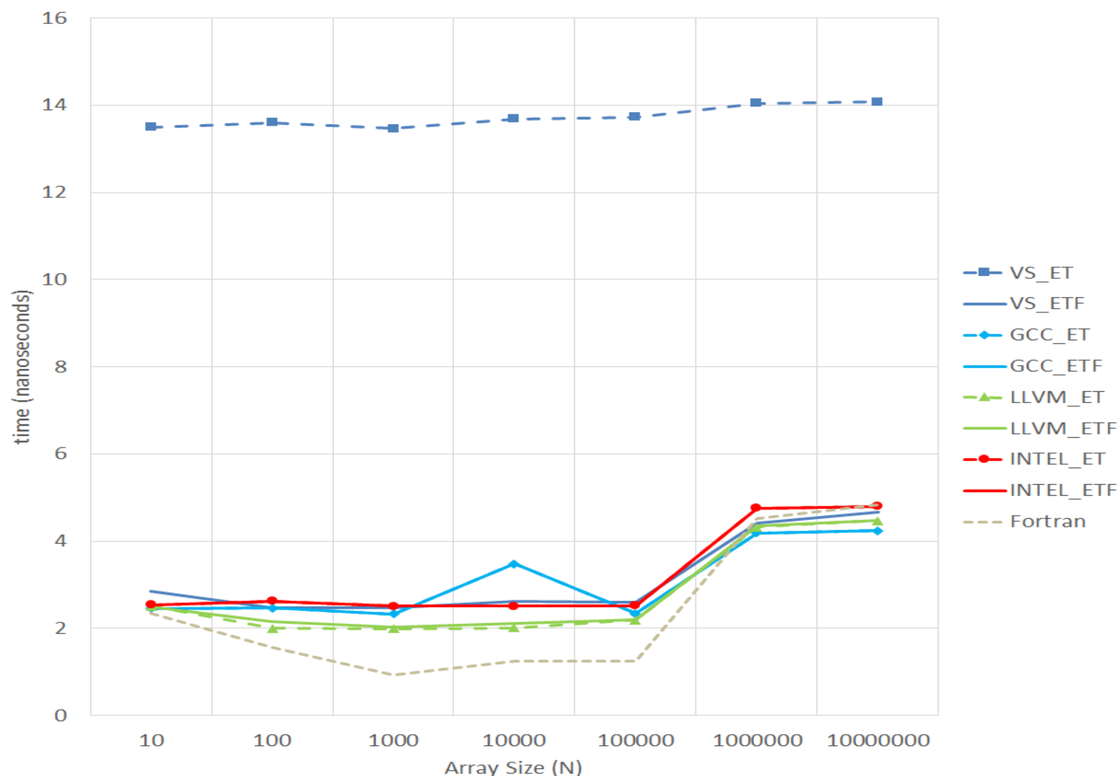
شکل ۶: زمان‌سنجی با مترجم‌های مختلف برای تابع $g(x.y.z.v) = xy + zv$
(روش ج: خطوط خط چین با نمادها، روش د: خطوط تو پُر)

۴-۳- مقایسه زمان ترجمه و حجم فایل اجرایی

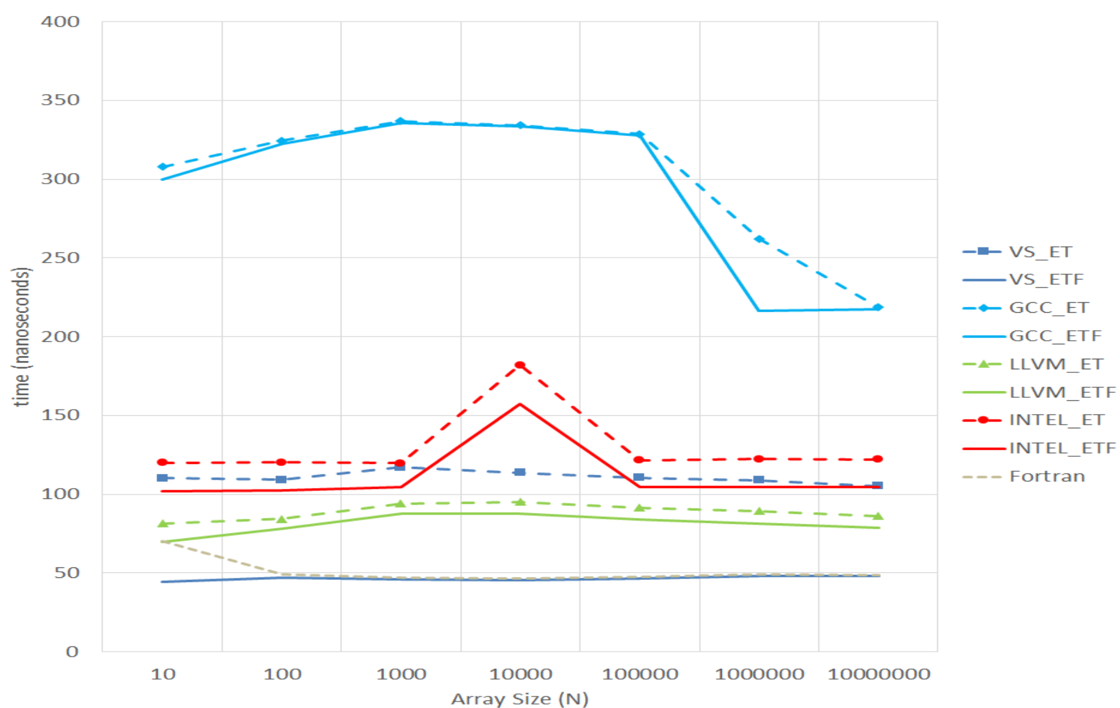
از آنجا که مترجم در برخورد با الگوها نیاز دارد کلاس متناظر با عبارت را تولید کند و نیز (اغلب) به‌طور خودکار، نوع پارامتر الگو را باید تشخیص دهد، انتظار می‌رود هرچه پیچیدگی کلاس استنتاج‌شده کمتر باشد، زمان ترجمه نیز کمتر شود. با الگوهای عبارت به روش مرسوم (روش ج) پیچیدگی کلاس نهایی به تعداد جملات موجود در عبارت بستگی دارد، در حالی که با الگوهای متغیر (روش د) پیچیدگی کلاس تنها به تعداد آرگومان‌های تابع بستگی دارد. برای مثال، با روش ج کلاس نهایی برای تابع $h(x.y.z.v)$ شامل ۱۳ کلاس MultiExpr و ۱۰ کلاس arr بوده، در حالی که با روش د کلاس نهایی به‌صورت MultiExpr<h,arr,arr,arr,arr> است. بنابراین انتظار می‌رود مترجم (حین ترجمه الگوها) زمان بیشتری برای روش ج صرف کند.

www.SID.ir

Archive of SID



شکل ۷: زمان‌سنجی با مترجم‌های مختلف برای تابع $f(x) = x + xx + xxx + xxxx$
(روش ج: خطوط خط چین با نمادها، روش د: خطوط تو پُر)



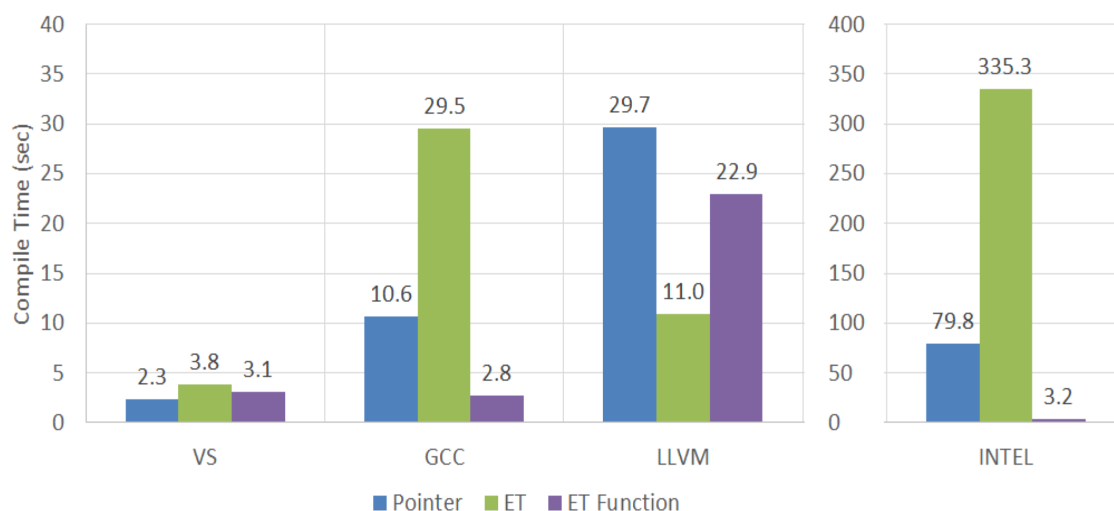
شکل ۸: زمان‌سنجی با مترجم‌های مختلف برای تابع $h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$
(روش ج: خطوط خط چین با نمادها، روش د: خطوط تو پُر)

www.SID.ir

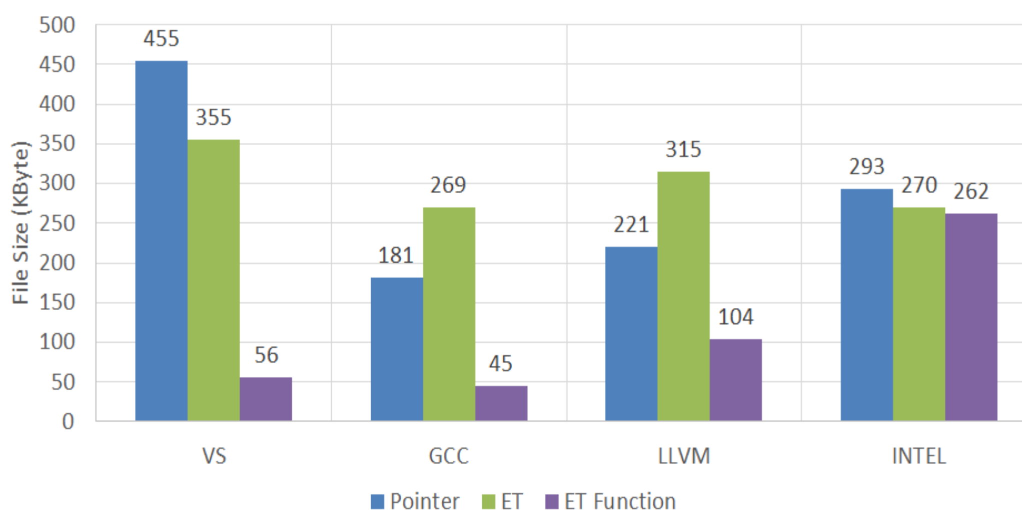
Archive of SID

برای تابع $h(x.y.z.v)$ زمان‌های ترجمه برای روش‌های الف، ج و د مقایسه می‌شود. برای اینکه زمان ترجمه قابل‌ملاحظه باشد، نیاز است تابع به تعداد زیاد (در اینجا هزار بار) تکثیر شود. شکل (۹) زمان‌های ترجمه را برای مترجم‌های مختلف نمایش می‌دهد. مشاهده می‌شود با مترجم‌های ویژوال استودیو، جی‌سی‌سی و اینتل، زمان ترجمه برای روش د کمتر از روش ج است و برای مترجم ال‌ال‌وی‌ام زمان ترجمه برای روش د بیشتر از روش ج. مترجم اینتل زمان زیادی صرف ترجمه روش ج و همچنین روش الف می‌کند؛ اما برای روش د زمان ترجمه بسیار کم است. برای مترجم جی‌سی‌سی نیز تا حدی این مورد مشاهده می‌شود.

در شکل (۱۰) مقایسه‌ای برای حجم فایل اجرایی تولیدشده صورت گرفته است. مشاهده می‌شود برای تمام مترجم‌ها کمترین حجم فایل، مربوط به روش د است. برای مترجم‌های ویژوال استودیو و جی‌سی‌سی حجم فایل با روش د به مقدار قابل‌ملاحظه‌ای کمتر از روش الف و ج است.



شکل ۹: زمان ترجمه با مترجم‌های مختلف برای تابع $h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$



شکل ۱۰: حجم فایل اجرایی با مترجم‌های مختلف برای تابع $h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$

www.SID.ir

Archive of SID

۴-۴- زمان‌سنجی با پردازنده گرافیکی

با استفاده از زبان برنامه‌نویسی کودا زمان‌سنجی با پردازنده گرافیکی صورت می‌گیرد. ازآنجاکه استفاده از عملگر ++ در این گونه محاسبات که به‌صورت موازی انجام می‌شود، وجهی ندارد، در اینجا تنها روش‌های الف، ب و د زمان‌سنجی می‌شوند. استفاده از نحو روش‌های ج و د همانند نحو پردازنده مرکزی بوده، اما برای روش الف نیاز به تعریف یک کِرنل جداگانه است. برای نمونه، تعریف تابع $g(x.y.z.v)$ به‌صورت زیر است:

```
__global__ void g(double *r, int size, double *a, double *b, double *c, double *d)
{
	int i = blockDim.x*blockIdx.x + threadIdx.x;
	if (i < size)
	{
		r[i] = a[i]*b[i] + c[i]*d[i]
	}
}
```

و نحوه فراخوانی آن به‌صورت زیر خواهد بود:

```
threadPerBlock = 256;
blockPerGrid = ceil( double(N) / threadPerBlock);
g<<<blockPerGrid, threadPerBlock >>>(R.a, N, A.a, B.a, C.a, D.a);
```

در شکل‌های (۱۱) تا (۱۳) نتایج زمان‌سنجی برای یک پردازنده گرافیکی نشان داده شده است. به علت ساختار سخت‌افزاری پردازنده‌های گرافیکی، بهتر است تعداد درایه‌ها مضربی از ۳۲ باشند، بنابراین تعداد درایه‌ها از$N=2^{10}$ تا $N=2^{23}$ در نظر گرفته شده‌اند. هرچند تعداد نخ‌های هر بلاک را می‌توان ۳۲ یا مضارب آن در نظر گرفت (البته تا سقف مجاز سخت‌افزار که ۱۰۲۴ است)، با این حال، مطابق تجربیات پیشین [۲۲،۲۳] برای پردازنده گرافیکی استفاده‌شده در مقاله حاضر، بهترین تعداد نخ‌ها ۱۲۸ یا ۲۵۶ است. بنابراین در تمام محاسبات، تعداد نخ‌های هر بلاک برابر ۲۵۶ در نظر گرفته می‌شود.

در شکل (۱۱) برای تابع $g(x.y.z.v)$ مشاهده می‌شود هر سه روش، کارایی یکسانی دارند. اما برای تابع یک‌متغیره $f(x)$ در شکل (۱۲) مشاهده می‌شود که کارایی استفاده از روش الگوهای عبارت مرسوم با افت کارایی مواجه است؛ درحالی‌که با روش معرفی‌شده در مقاله حاضر (روش د) کارایی همانند استفاده از آرایه‌ها است. برای تابع $h(x.y.z.v)$ در شکل (۱۳) مشاهده می‌شود هر سه روش، عملکرد مشابهی دارند و روش د کمی بهتر عمل می‌کند. در هر سه شکل مشاهده می‌شود که برای تعداد درایه‌های کم، پردازنده گرافیکی عملکرد مناسبی ندارد؛ زیرا محاسبات بخشی از زمان اجرای کِرنل است و بارگذاری کِرنل نیز زمانی به خود اختصاص می‌دهد که البته بیشتر از زمان بارگذاری توابع اجراشده روی پرازنده مرکزی است. با افزایش تعداد درایه‌ها نسبت زمان بارگذاری کِرنل به زمان محاسبات کمتر می‌شود و در نتیجه، زمان متناظر برای یک درایه در هر سه شکل به یک عدد ثابت می‌رسد.

شکل (۱۴) افزایش سرعت حاصل‌شده با استفاده از پردازنده گرافیکی برای روش د را برای هر سه تابع نشان می‌دهد. برای تعداد درایه کم، مطابق انتظار افزایش سرعت کمی مشاهده می‌شود که دلیل آن در بالا ذکر شد. با افزایش تعداد درایه‌ها برای تابع $g(x.y.z.v)$ و $f(x)$ به افزایش سرعتی حدود ۱۵ برابر و در نهایت، به حدود ۳۰ برابر می‌رسد. برای تابع $h(x.y.z.v)$ که برای تمام تعداد درایه‌ها بیشترین افزایش سرعت را دارد، این مقدار به نزدیک ۵۰ برابر نیز می‌رسد. البته اینکه بیشترین افزایش سرعت مربوط به بیشترین تعداد است (سمت راست شکل (۱۴))، به علت اُفت کارایی پردازنده مرکزی است که در سمت راست شکل‌های (۳) تا (۵) نیز مشاهده شد. در مجموع، طراحی ارائه‌شده (روش د) برای استفاده از قابلیت‌های پردازنده‌های گرافیکی نیز مناسب است.

۴-۵- حل عددی معادلات دینامیک گاز اویلر

در این قسمت، برای نشان دادن کاربردی از روش پیشنهادی در شبیه‌سازی‌های عددی، مسئله لوله-شوک را در نظر می‌گیریم [۲۴]. در این مسئله معادلات حاکم، معادلات دینامیک گاز اویلر یک‌بُعدی‌اند. این معادلات حاکم بر سیالات‌تراکم‌پذیر غیرلزج هستند:

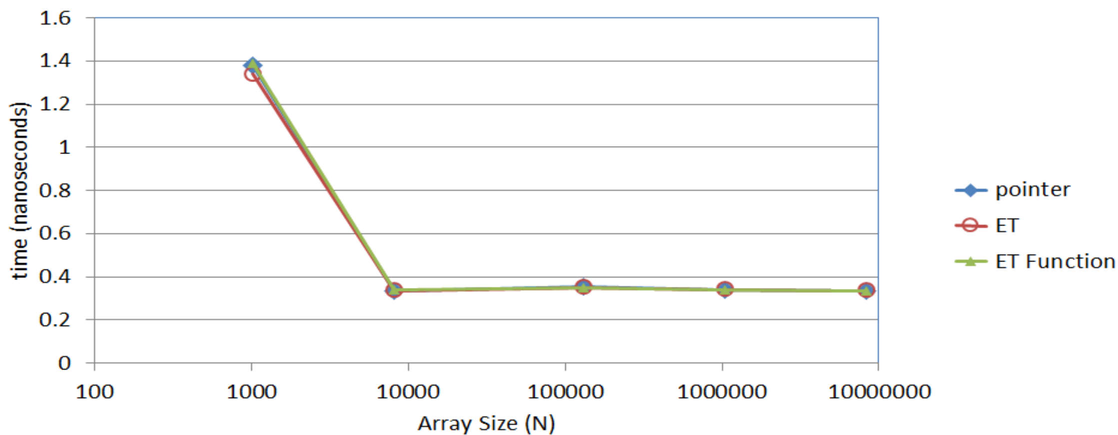
$$U_t + F_x = 0\,, U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E+p)u \end{pmatrix}$$

$$E = \rho\left(e + \frac{u^2}{2}\right), p = \rho e(\gamma - 1), \gamma = 1.4$$

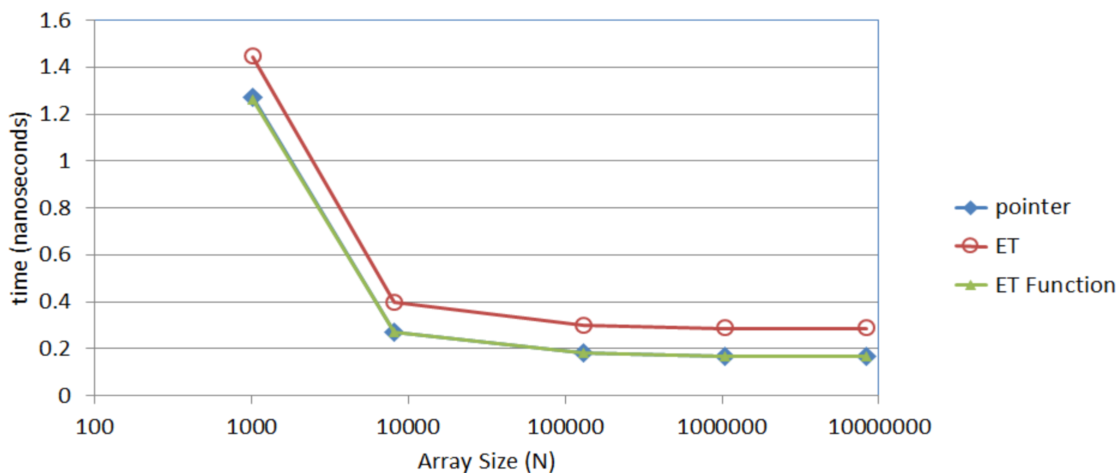
که در آن، ρ چگالی، u سرعت، p فشار، e انرژی دورنی و γ نسبت گرماهای ویژه و همچنین U بردار متغیرهای پایستار و F بردار شار است.

www.SID.ir

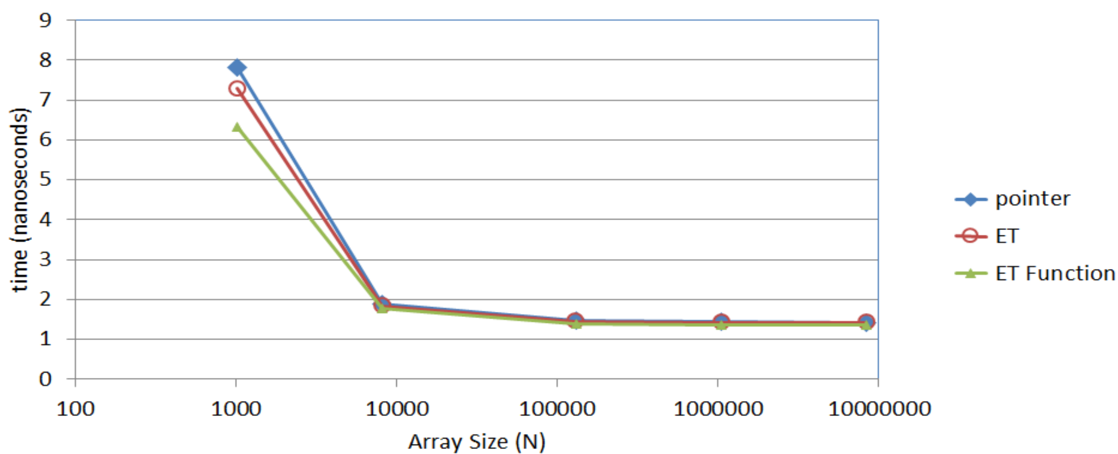
Archive of SID



شکل ۱۱: زمان‌سنجی با پردازنده گرافیکی برای تابع $g(x.y.z.v) = xy + zv$



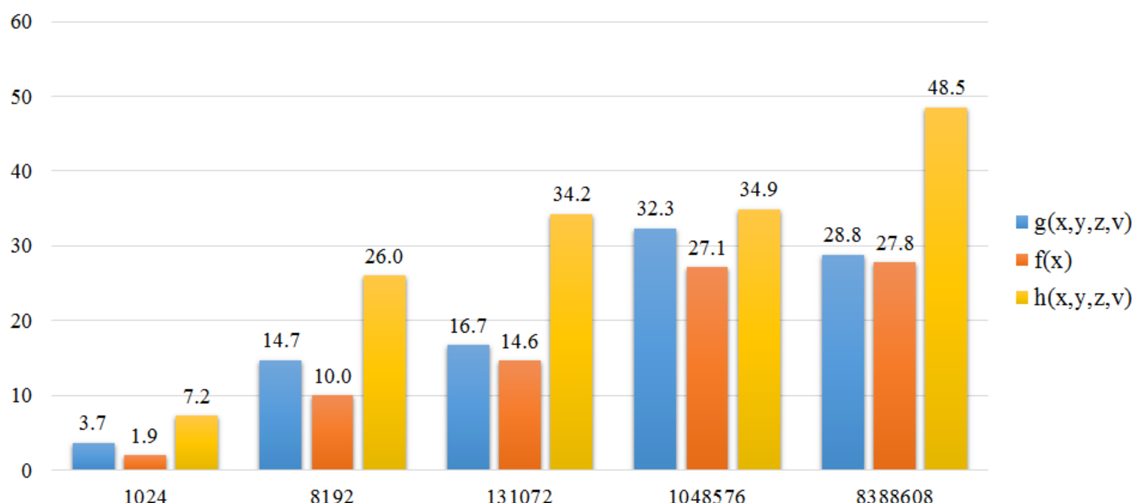
شکل ۱۲: زمان‌سنجی با پردازنده گرافیکی برای تابع $f(x) = x + xx + xxx + xxxx$



شکل ۱۳: زمان‌سنجی با پردازنده گرافیکی برای تابع $h(x.y.z.v) = \sin(x) - \cos(xy) + \sin(xyz) - \cos(xyzv)$

www.SID.ir

Archive of SID



شکل ۱۴: نسبت زمان اجرا پردانده مرکزی به پردازنده گرافیکی برای روش د

در این مسئله یک‌جداره نازک، هوای درون یک مخزن را به دو قسمت تقسیم کرده است که فشار هوا در دو سمت جداره متفاوت است. در یک لحظه، جداره به‌سرعت برداشته می‌شود که اختلاف فشار، موجب حرکت گاز از سمت پُرفشار به سمت کم‌فشار می‌شود. در اینجا طول مخزن برابر ۱۰ و جداره نازک در وسط آن قرار دارد. شرایط اولیه به‌صورت زیر است:

$$(\rho,u,p)=\begin{cases}(1.0, 0.0, 1.0) & x<5\\(0.125, 0.0, 0.1) & x\geq 5\end{cases}$$

در حل عددی معادلات نیاز است که بردار شار F به شارهای مثبت و منفی تجزیه شود:

$$F=F^{+}+F^{-}$$

پس از تجزیه شار رابطه گسسته برای حل عددی به‌صورت زیر است:

$$U_i^{n+1}=U_i^n-\frac{\Delta t}{\Delta x}\left(F_i^{+}-F_{i-1}^{+}+F_{i+1}^{-}-F_i^{-}\right)$$

برای هریک از متغیرها نیاز به تعریف یک آرایه است:

```
arr rho,u,p,E;
arr U[3],F[3],Fp[3],Fm[3],UNEW[3];
```

روش‌های مختلفی برای تجزیه شار وجود دارد که در اینجا روش تجزیه شار ون‌لیر[۲۴] انتخاب می‌شود.

$$F^{+}=F=\begin{pmatrix}\rho u\\ \rho u^2+p\\ (E+p)u\end{pmatrix},\qquad M>1$$

$$F^{+}=\begin{pmatrix}0\\0\\0\end{pmatrix},\qquad M<-1$$

$$F^{+}=\frac{\rho a(1+M)^2}{4}\begin{pmatrix}1\\ a\left(\frac{\gamma-1}{\gamma}M+\frac{2}{\gamma}\right)\\ a^2\left(\frac{\gamma-1}{\gamma}M+\frac{2}{\gamma}\right)^2\frac{\gamma^2}{2(\gamma^2-1)}\end{pmatrix}$$

$$-1\leq M\leq 1$$

$$M=\frac{u}{a},\qquad a=\sqrt{\gamma\frac{p}{\rho}},F^{-}=F-F^{+}$$

که در آن، a سرعت صوت و M عدد ماخ است. درحقیقت، تجزیه شار، یک تابع چندضابطه‌ای بوده که وابسته به عدد ماخ M است. برای هریک از مؤلفه‌های F^{+} نیاز به تعریف یک تابع خواهد بود. شایان ذکر است نه‌تنها به‌علت طولانی بودن روابط، بلکه به‌علت چندضابطه‌ای بودن آن‌ها نیاز به تعریف تابع وجود دارد. پیاده‌سازی این توابع به‌صورت زیر است:

```
struct PositiveFlux_0 : public MultiVarFun <
PositiveFlux_0 >
{
        static double apply(real rho, real u,
real p)
        {
        double a = sqrt(gam*p/rho);
        double M = u/a;

        if(M>1)
                return rho*u;
        else if(M<-1)
                return 0.0;
        else
                return
0.25*rho*a*(1.0+M)*(1.0+M);
```


www.SID.ir

Archive of SID

```
            }
};

struct PositiveFlux_1 : public MultiVarFun <
PositiveFlux_1 >
{
            static double apply(real rho, real u,
real p)
            {
            double a = sqrt(gam*p/rho);
            double M = u/a;
            if(M>1)
                        return rho*u*u+p;
            else if(M<-1)
                        return 0.0;
            else
                        {
                        double s =
0.25*rho*a*(1.0+M)*(1.0+M);
                        double q = a*((gam-
1.0)*M+2.0)/gam;
                        return s*q;
                        }
            }
};
struct PositiveFlux_2 : public MultiVarFun <
PositiveFlux_2>
{
            static double apply(real rho, real u,
real p)
            {
            double a = sqrt(gam*p/rho);
            double M = u/a;
            double E = p/(gam-1)+0.5*rho*u*u;

            if(M>1)
                        return (E+p)*u;
            else if(M<-1)
                        return 0.0;
            else
                        {
                        double s =
0.25*rho*a*(1.0+M)*(1.0+M);
                        double q = a*((gam-
1.0)*M+2.0)/gam;
            return 0.5*s*q*q*gam*gam/
              gam*gam-1.0);
                        }           }};
```

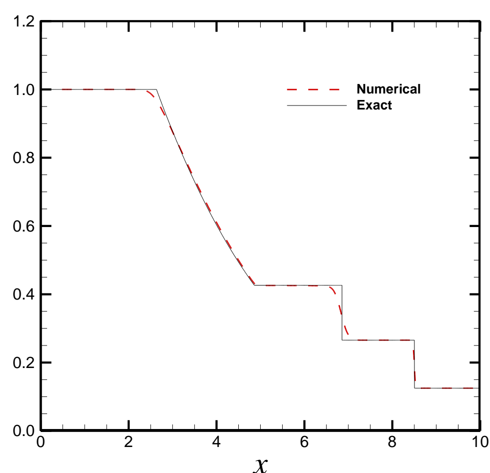
با تعریف توابع فوق و نیز چند تابع دیگر، محاسبات اصلی برای حل معادلات به‌صورت زیر انجام می‌شود:

```
for (int k = 0; k < maximumTimeStep; k++)
  {
```

```
        rho = U[0];
        u = U[1]/U[0];
        E = U[2];
        p = EnergyToPressure::fun(rho,u,E);
        F[0] = U[1];
        F[1] = rho*u*u+p;
        F[2] = (E+p)*u;
        Fp[0] = PositiveFlux_0::fun(rho,u,p);
        Fp[1] = PositiveFlux_1::fun(rho,u,p);
        Fp[2] = PositiveFlux_2::fun(rho,u,p);
        Fm[0] = F[0] - Fp[0];
        Fm[1] = F[1] - Fp[1];
        Fm[2] = F[2] - Fp[2];
        for(int m=0;m<3;++m)
        {
          for (int i = 1; i < N - 1; i++)
          {
            UNEW[m][i] = U[m][i] –
                                dt / dx*(Fp[m][i] -
Fp[m][i - 1] +
                                Fm[m][i + 1] -
Fm[m][i]);
          }
        }

        for(int m=0;m<3;++m)
            U[m]=UNEW[m];
    }
```

مشاهده می‌شود که با الگوهای عبارت و روش پیشنهادی برنامه نوشته‌شده کوتاه و کاملاً خوانا است. شکل (۱۵) نتایج حل عددی را با نتایج حل دقیق در زمان $t = 2$ مقایسه می‌کند.



شکل ۱۵: مقایسه حل عددی با حل دقیق برای مسأله لوله شوک

www.SID.ir

Archive of SID

۵- نتیجه‌گیری

در این مقاله، روشی کارآمد برای تعریف توابع چندمتغیره جهت استفاده برای عملیات روی آرایه‌ها در الگوهای عبارت معرفی شد. با این روش، علاوه بر کارایی، سادگی تعریف توابع برای کاربر حفظ شد. برای بررسی کارایی روش، سه تابع مختلف با نحوهای گوناگون در سی‌پلاس‌پلاس پیاده‌سازی و زمان‌های اجرا با مترجم ویژوال استودیو برای آرایه‌ها با تعداد درایه‌های مختلف مقایسه گردید. همچنین برای مقایسه بهتر، زمان‌های اجرا با فرترن نیز آورده شد. زمان‌سنجی‌ها نشان داد که برای هر سه تابع روش معرفی‌شده برای هر تعداد درایه یا بهترین کارایی را دارد یا به بهترین کارایی نزدیک است.

زمان‌سنجی با مترجم‌های جی‌سی‌سی، اِل‌اِل‌وی‌اِم و اینتل نیز صورت گرفت. برای دو تابع زمان‌های اجرا با الگوهای عبارت مرسوم تفاوتی با روش پیشنهادی نداشت، ولی برای تابع سوم روش پیشنهادی زمان اجرای کمتری داشت.

از نظر زمان ترجمه و نیز حجم فایل اجرایی روش پیشنهادی با سایر روش‌ها مقایسه شد. مشاهده گردید که روش پیشنهادی نسبت به الگوهای عبارت مرسوم با تمام مترجم‌ها (غیر از ال‌ال‌وی‌ام) زمان ترجمه کمتری نیاز دارد. همچنین با تمام مترجم‌ها حجم فایل اجرایی با روش پیشنهای کمترین حجم را دارد.

همچنین نشان داده شد که استفاده از روش مذکور روی پردازنده‌های گرافیکی به‌راحتی صورت می‌گیرد. نتایج زمان‌سنجی مؤید این بود که تفاوتی بین نحوهای گوناگون روی پردازنده گرافیکی وجود ندارد و روش معرفی‌شده اندکی کارایی بهتری دارد. بنابراین، روش ارائه‌شده می‌تواند روشی مناسب برای افزایش قابلیت‌های کتابخانه‌های عددی، هم برای پردازنده‌های مرکزی و هم برای پردازنده‌های گرافیکی باشد.

تقدیر و تشکر

ازحمایت مالی دانشگاه تهران و بنیاد ملی نخبگان از این تحقیق در قالب طرح پژوهشی شماره ۰۲/۱/۲۸۷۴۵ قدردانی می‌شود.

واژه نامه

اُپن‌فوم
OpenFoam, 2

الگوها
Templates, 3

الگوهای عبارت
Expression templates, 4

الگوهای متغیر
Variadic templates, 4

اُورچِر
Overture, 2

برنامه‌نویسی شیءگرا
Object-Oriented Programming, 3

بلتیز پلاس‌پلاس
Blitz++, 3

بوست یوبلَس
Boost uBlas, 3

پردازنده‌های گرافیکی
Graphics processing units, 4

پوما
Pooma, 3

دیل ۲
deal.II, 2

دیون
Dune, 2

سربارگذاری عملگرها
Operator overloading, 5

فرابرنامه‌نویسی
Metaprogramming, 3

فِنیکس
FEniCS, 2

کتابخانه الگوی استاندارد
Standard Template Library (STL), 3

کِرنل
Kernel, 19

ماکرو
Macro, 4

متن-باز
open-source, 2

نحو
Syntax, 4

نخ‌های هر بلاک
Threads per block, 19

نکتارپلاس‌پلاس
Nektar++, 2

www.SID.ir

Archive of SID

مراجع

[1] https://openfoam.org.

[2] H. Jasak, A. Jemcov and Z. Tukovic, “Openfoam: A C++ library for complex physics simulations”, International Workshop on Coupled Methods in Numerical Dynamics, IUC, Dubrovnik, Croatia, September 2007, PP. 19-21.

[3] http://www.overtureframework.org.

[4] W.D. Henshaw, “Overture: An object-oriented system for solving PDEs in moving geometries on overlapping grids”, First AFOSR Conference on Dynamic Motion CFD, 1996.

[5] W. Bangerth, R. Hartmann and G. Kanschat, "deal.II – a general purpose object oriented finite element library", ACM Transactions on Mathematical Software, Vol. 33, Issue 4, Article 24, 2007.

[6] W. Bangerth, D. Davydov, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin and D. Wells, "The deal.II library, version 8.4", Journal of Numerical Mathematics,Vol. 24, 2016, pp. 135-141.

[7] M. Blatt and P. Bastian, "The iterative solver template library", Applied Parallel Computing, State of the Art in Scientific Computing, Vol. 4699 of Lecture Notes in Computer Science, Springer, 2007.

[8] M. Blatt and P. Bastian, "On the generic parallelisation of iterative solvers for the finite element method", International Journal of Computational Science and Engineering, Vol. 4, Issue 1, 2008, pp. 56–69.

[9] http://www.dune-project.org.

[10] A. Logg, K.A. Mardal, G.N. Wells and et al., Automated Solution of Differential Equations by the Finite Element Method, Lecture Notes in Computational Science and Engineering, Springer, 2012.

[11] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, 'The fenics project version 1.5", Archive of Numerical Software, Vol. 3, No. 100. 2015.

[12] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied,C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby and S.J. Sherwin, "Nektar++: An open-source spectral/element framework", Computer Physics Communications, Vol. 192, 2015, pp. 205–219.

[13] D. Abrahams and A. Gurtovoy, C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond, Addison-Wesley, 2005.

[14] http://www.nongnu.org/freepooma.

[15] http://www.boost.org/doc/libs/1610/libs/numeric/ublas/doc.

[16] http://blitz.sourceforge.net.

[17] H. Schildt, C++ The Complete Reference, 5thEdition, McGraw-Hill Education, 2012.

[18] M. Hemmat Esfe and S. Sadodin, "Flow Behavior and Thermal Performance of Double Lid-Driven Cavity Subjected to Nanofluid with Variable Properties", Journal of Modeling in Engineering, Vol. 10, Issue 30, 2012, pp. 43-60.

www.SID.ir

Archive of SID

[19] A.A. Abbasian Arani, A. Aghaee and H. Ehteram, "Numerical Investigation of Brownian Motion Effect on Nanofluid Mixed Convection on Enclosure with a Hot Central Heat Source", Journal of Modeling in Engineering, Vol. 11. Issue 34, 2013, pp. 15-29.

[20] S. Sadodin, M. Hemmat Esfe and M.J. Noroozi, "Numerical Simulation of Mixed Convection of Fluid Flow and Heat Transfer within Car Radiator with an Inside Hot Obstacle Filled with Nanofluid", Journal of Modeling in Engineering, Vol. 9, Issue 25, 2011, pp. 33-46.

[21] K. Iglberger, G. Hager, J. Treibig and U. Rde, "Expression templates revisited: A performance analysis of current methodologies", SIAM Journal on Scientific Computing, Vol. 34, Issue 2, 2012, pp. C42–C69.

[22] V. Esfahanian, H. Mahmoodi Darian and S.M.I. Gohari,"Assessment of weno schemes for numerical simulation of some hyperbolic equations using gpu", Computers and Fluids, Vol. 80, 2013, pp. 260–268.

[23] H. Mahmoodi Darian and V. Esfahanian, "Assessment of weno schemes for multi-dimensional euler equations using gpu", International Journal for Numerical Methods in Fluids, Vol. 76, Issue 12, 2014, pp. 961–981.

[24] K.A. Hoffmann, and S.T. Chiang, Computational Fluid Dynamics, 4th Edition, Engineering Education System, 2000.

www.SID.ir